Logical Methods in Computer Science
Vol. 5 (2:3) 2009, pp. 1–29
www.lmcs-online.org

Submitted Mar. 8, 2008
Published Apr. 22, 2009

A GENERIC FRAMEWORK FOR REASONING ABOUT DYNAMIC NETWORKS OF INFINITE-STATE PROCESSES [*]

AHMED BOUAJJANI, CEZARA DRĂGOI, CONSTANTIN ENEA, YAN JURSKI, AND MIHAELA SIGHIREANU

LIAFA, University Paris Diderot and CNRS, Case 7014, 75205 Paris Cedex 13, France.
e-mail address: {abou,cezarad,cenea,jurski,sighirea}@liafa.jussieu.fr

Abstract. We propose a framework for reasoning about unbounded dynamic networks of infinite-state processes. We propose Constrained Petri Nets (CPN) as generic models for these networks. They can be seen as Petri nets where tokens (representing occurrences of processes) are colored by values over some potentially infinite data domain such as integers, reals, etc. Furthermore, we define a logic, called CML (colored markings logic), for the description of CPN configurations. CML is a first-order logic over tokens allowing to reason about their locations and their colors. Both CPNs and CML are parametrized by a color logic allowing to express constraints on the colors (data) associated with tokens.

We investigate the decidability of the satisfiability problem of CML and its applications in the verification of CPNs. We identify a fragment of CML for which the satisfiability problem is decidable (whenever it is the case for the underlying color logic), and which is closed under the computations of post and pre images for CPNs. These results can be used for several kinds of analysis such as invariance checking, pre-post condition reasoning, and bounded reachability analysis.

1. Introduction

The verification of software systems requires in general the consideration of infinite-state models. The sources of infinity in software models are multiple. One of them is the manipulation of variables and data structures ranging over infinite domains (such as integers, reals, arrays, etc). Another source of infinity is the fact that the number of processes running in parallel in the system can be either a parameter (fixed but arbitrarily large), or it can be dynamically changing due to process creation. While the verification of parameterized systems requires reasoning uniformly about the infinite family of (static) networks corresponding to any possible number of processes, the verification of dynamic

1998 ACM Subject Classification: E.1, F.3.1, F.4.1, F.4.3, I.2.2.

Key words and phrases: dynamic networks, colored Petri nets, first-order logic, verification.

[*] A shorter version of this paper has been published in the Proceedings of TACAS 2007, LNCS 4424. This work is partially supported by the French ANR project AVERISS.

DOI:10.2168/LMCS-5 (2:3) 2009

systems requires reasoning about the infinite number of all possible dynamically changing network configurations.

There are many works and several approaches on the verification of infinite-state systems taking into account either the aspects related to infinite data domains, or the aspects related to unbounded network structures due to parametrization or dynamic creation of processes. Concerning systems with data manipulation, a lot of work has been devoted to the verification of, for instance, finite-structure systems with unbounded counters, clocks, stacks, queues, etc. (see, e.g., [AvJT96, BEM97, WB98, Boi99, AAB00, FS01, FL02]). On the other hand, a lot of work has been done for the verification of parameterized and dynamic networks of Boolean (or finite-data domain) processes, proposing either exact model-checking and reachability analysis techniques for specific classes of systems (such as broadcast protocols, multithreaded programs, etc) [EN98, EFM99, DRB02, BT05, BMOT05], or generic algorithmic techniques (which can be approximate, or not guaranteed to terminate) such as network invariants-based approaches [WL89, CGJ97], and (abstract) regular model checking [BJNT00, Bou01, AJNS04, BHV04]. However, only few works consider both infinite data manipulation and parametric/dynamic network structures (see the paragraph on related work).

In this paper, we propose a generic framework for reasoning about parameterized and dynamic networks of concurrent processes which can manipulate (local and global) variables over infinite data domains. Our framework is parameterized by a data domain and a first-order theory on it (e.g., Presburger arithmetics on natural numbers). It consists of (1) expressive models allowing to cover a wide class of systems, and (2) a logic allowing to specify and to reason about the configurations of these models.

The models we propose are called Constrained Petri Nets (CPN for short). They are based on (place/transition) Petri nets where tokens are colored by data values. Intuitively, tokens represent different occurrences of processes, and places are associated with control locations and contain tokens corresponding to processes which are at a same control location. Since processes can manipulate local variables, each token (process occurrence) has several colors corresponding to the values of these variables. Then, configurations of our models are markings where each place contains a set of colored tokens, and transitions modify the markings as usual by removing tokens from some places and creating new ones in some other places. Transitions are guarded by constraints on the colors of tokens before and after firing the transition. We show that CPNs allow to model various aspects such as unbounded dynamic creation of processes, manipulation of local and global variables over unbounded domains such as integers, synchronization, communication through shared variables, locks, etc.

The logic we propose for specifying configurations of CPNs is called Colored Markings Logic (CML for short). It is a first order logic over tokens and their colors. It allows to reason about the presence of tokens in places, and also about the relations between the colors of these tokens. The logic CML is parameterized by a first order logic over the color domain allowing to express constraints on tokens.

We investigate the decidability of the satisfiability problem of CML and its applications in verification of CPNs. While the logic is decidable for finite color domains (such as booleans), we show that, unfortunately, the satisfiability problem of this logic becomes undecidable as soon as we consider the color domain to be the set of natural numbers with the usual ordering relation (and without any arithmetical operations). We prove that this undecidability result holds already for the fragment $\forall^*\exists^*$ of the logic (in the alternation hierarchy of the quantifiers over token variables) with this color domain.

On the other hand, we prove that the satisfiability problem is decidable for the fragment $\exists^*\forall^*$ of CML whenever the underlying color logic has a decidable satisfiability problem, e.g., Presburger arithmetics, the first-order logic of addition and multiplication over reals, etc. Moreover, we prove that the fragment $\exists^*\forall^*$ of CML is effectively closed under post and pre image computations (i.e., computation of immediate successors and immediate predecessors) for CPNs where all transition guards are also in $\exists^*\forall^*$. We show also that the same closure results hold when we consider the fragment $\exists^*$ instead of $\exists^*\forall^*$.

These generic decidability and closure results can be applied in the verification of CPN models following different approaches such as pre-post condition (Hoare triples based) reasoning, bounded reachability analysis, and inductive invariant checking. More precisely, we derive from our results mentioned above that (1) checking whether starting from a $\exists^*\forall^*$ pre-condition, a $\forall^*\exists^*$ condition holds after the execution of a transition is decidable, that (2) the bounded reachability problem between two $\exists^*\forall^*$ definable sets is decidable, and that (3) checking whether a formula defines an inductive invariant is decidable for Boolean combinations of $\exists^*$ formulas.

These results can be used to deal with non trivial examples of systems. Indeed, in many cases, program invariants and the assertions needed to establish them fall in the considered fragments of our logic. We illustrate this by carrying out in our framework the verification of several parameterized systems (including the examples usually considered in the literature such as the Bakery mutual exclusion protocol [Lam74]). In particular, we provide an inductive proof of correctness for the parametric version of the Reader-Writer lock system introduced in [FFQ02]. Flanagan et al. give a proof of this case study for the case of one reader and one writer. We consider here an arbitrarily large number of reader and writer processes and carry out (for the first time, to our knowledge) its verification by inductive invariant checking. We provide experimental results obtained for these examples using a prototype tool we have implemented based on our decision and verification procedures.

Related work: The use of unbounded Petri nets as models for parameterized networks of processes has been proposed in many existing works such as [GS92, EN98, DRB02]. However, these works consider networks of *finite-state* processes and do not address the issue of manipulating infinite data domains. The extension of this idea to networks of infinite-state processes has been addressed only in very few works [AJ98, Del01, BD02, AD06]. In [AJ98], Abdulla and Jonsson consider the case of networks of 1-clock timed systems and show, using the theory of well-structured systems and well quasi orderings [AvJT96, FS01], that the verification problem for a class of safety properties is decidable. Their approach has been extended in [Del01, BD02] to a particular class of multiset rewrite systems with constraints (see also [AD06] for recent developments of this approach). Our modeling framework is actually inspired by these works. However, while they address the issue of deciding the verification problem of safety properties (by reduction to the coverability problem) for specific classes of systems, we consider in our work a general framework, allowing to deal in a generic way with various classes of systems, where the user can express assertions about the configurations of the system, and check automatically that they hold (using post-pre reasoning and inductive invariant checking) or that they do not hold (using bounded reachability analysis). Our framework allows to reason automatically about systems which are beyond the scope of the techniques proposed in [AJ98, Del01, BD02, AD06] such as, for instance, the parameterized Reader-Writer lock system presented in this paper.

In parallel to our work, Abdulla et al. developed in [ADHR07, AHDR08] abstract backward reachability analysis for a restricted class of constrained multiset rewrite systems. Basically, they consider constraints which are boolean combinations of universally quantified formulas, where data constraints are in the particular class of existentially quantified gap-order constraints. The abstraction they consider consists in taking after each pre-image computation the upward closure of the obtained set. This helps termination of the iterative computation and yields an upper-approximation of the backward reachability set. However, the used abstract analysis can be too imprecise for some systems. Our approach allows in contrast to carry out pre-post reasoning, invariance checking, as well as bounded analysis, for a larger class of systems. Techniques like those used in [ADHR07, AHDR08] could be integrated into our framework in the future in order to discover (local) invariants automatically.

In a series of papers, Pnueli et al. developed an approach for the verification of parameterized systems combining abstraction and proof techniques (see, e.g., [APR$^+$01]). This is probably one of the most advanced existing approaches allowing to deal with unbounded networks of infinite-state processes. We propose here a different framework for reasoning about these systems. In [APR$^+$01], the authors consider a logic on (parametric-bound) *arrays* of integers, and they identify a fragment of this logic for which the satisfiability problem is decidable. In this fragment, they restrict the shape of the formula (quantification over indices) to formulas in the fragment $\exists^*\forall^*$ similarly to what we do, and also the class of used arithmetical constraints on indices and on the associated values. In a recent work by Bradley et al. [BMS06b], the satisfiability problem of the logic of unbounded arrays with any kind of elements values is investigated and the authors provide a new decidable fragment, which is incomparable to the one defined in [APR$^+$01], but again which imposes similar restrictions on the quantifiers alternation in the formulas, and on the kind of constraints on indices that can be used. In contrast with these works, we consider a logic on *multisets* of elements with any kind of associated data values, provided that the used theory on the data domain is decidable. For instance, we can use in our logic general Presburger constraints whereas [APR$^+$01] allows limited classes of constraints. On the other hand, we cannot specify faithfully unbounded arrays in our decidable fragment because formulas of the form $\forall^*\exists^*$ are needed to express that every non extremal element has a successor/predecessor. Nevertheless, for the verification of safety properties and invariant checking, expressing this fact is not necessary, and therefore, it is possible to handle (model and verify) in our framework all usual examples of parameterized systems (such as mutual exclusion protocols) considered in the works cited above.

Let us finally mention that there are recent works on logics (first-order logics, or temporal logics) over finite/infinite structures (words or trees) over infinite alphabets (which can be considered as abstract infinite data domains) [BMS$^+$06a, BDM$^+$06, DL06]. The obtained positive results so far concern logics with very limited data domain (basically infinite sets with only equality, or sometimes with an ordering relation), and are based on reduction to complex problems such as reachability in Petri nets.

2. Colored Markings Logic

2.1. **Preliminaries.** Consider an enumerable set of *tokens* and let us identify this set with the set of natural numbers $\mathbb{N}$. Intuitively, tokens represent occurrences of (parallel) processes. We assume that tokens may have colors corresponding for instance to data values attached to the corresponding processes. We consider that each token has N colors, for some fixed natural number $N > 0$. Let $\mathbb{C}$ be a (potentially infinite) *token color domain*. Examples of color domains are the set of natural numbers $\mathbb{N}$ and the set of real numbers $\mathbb{R}$. Also, we consider that tokens can be located at *places*. Let $\mathbb{P}$ be a finite set of such places. Intuitively, places represent control locations of processes. A N-dim *colored marking* is a mapping $M \in [\mathbb{N} \to (\mathbb{P} \cup \{\bot\}) \times \mathbb{C}^N]$ which associates with each token its place (if it is defined, or $\bot$ otherwise) and the values of its colors.

Let M be a N-dim colored marking, let $t \in \mathbb{N}$ be a token, and let $M(t) = (p, c_1, \ldots, c_N) \in (\mathbb{P} \cup \{\bot\}) \times \mathbb{C}^N$. Then, we consider that $\mathit{place}_M(t)$ denotes the element p, that $\mathit{color}_M(t)$ denotes the vector $(c_1, \ldots, c_N)$, and that for every $k \in \{1, \ldots, N\}$, $\mathit{color}_{M,k}(t)$ denotes the element c_k. We omit the subscript M when it is clear from the context.

2.2. **Colored Markings Logic (CML).** The logic CML is parameterized by a (first-order) logic on the considered token color domain $\mathbb{C}$, $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$, i.e., by the set of operations Ω and the set of basic predicates (relations) Ξ allowed on $\mathbb{C}$. In the sequel, we omit all or some of the parameters of CML when their specification is not necessary.

Let T be a set of *token variables* ranging over $\mathbb{N}$ (set of tokens) and let C be a set of *color variables* ranging over $\mathbb{C}$, and assume that $T \cap C = \emptyset$. Then, the set of terms of $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$ (called *token color terms*) is given by the grammar:

$$t ::= z \mid \delta_k(x) \mid o(t_1, \ldots, t_n)$$

where $z \in C$, $k \in \{1, \ldots, N\}$, $x \in T$, and $o \in \Omega$. Intuitively, the term $\delta_k(x)$ represents the kth color (data value) attached to the token associated with the token variable x. We denote by $\equiv$ the syntactic equality relation on terms.

The set of *formulas* of $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$ is given by:

$$\varphi ::= \mathit{true} \mid x = y \mid p(x) \mid r(t_1, \ldots, t_m) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists z.\ \varphi \mid \exists x.\ \varphi$$

where $x, y \in T$, $z \in C$, $p \in \mathbb{P} \cup \{\bot\}$, $r \in \Xi$. As usual, *false* and the boolean connectives such as conjunction ($\wedge$) and implication ($\Rightarrow$), and universal quantification ($\forall$) can be defined in terms of *true*, $\neg$, $\vee$, and $\exists$. We also use $\exists x \in p.\ \varphi$ (resp. $\forall x \in p.\ \varphi$) as an abbreviation of the formula $\exists x.\ p(x) \wedge \varphi$ (resp. $\forall x.\ p(x) \Rightarrow \varphi$).

The notions of free/bound occurrences of variables in formulas and the notions of closed/open formulas are defined as usual in first-order logics. Given a formula φ, the set of free variables in φ is denoted $FV(\varphi)$. In the sequel, we assume w.l.o.g. that in every formula, each variable is quantified at most once.

We define a satisfaction relation between colored markings and CML formulas. For that, we need first to define the semantics of CML terms. Given valuations $\theta \in [T \to \mathbb{N}]$, $\nu \in [C \to \mathbb{C}]$, and a colored marking M, we define a mapping $\langle\!\langle \cdot \rangle\!\rangle_{M,\theta,\nu}$ which associates with each color term a value in $\mathbb{C}$:

$$\begin{array}{rcl} \langle\!\langle z \rangle\!\rangle_{M,\theta,\nu} & = & \nu(z) \\ \langle\!\langle \delta_k(x) \rangle\!\rangle_{M,\theta,\nu} & = & \mathit{color}_{M,k}(\theta(x)) \\ \langle\!\langle o(t_1,\ldots,t_n) \rangle\!\rangle_{M,\theta,\nu} & = & o(\langle\!\langle t_1 \rangle\!\rangle_{M,\theta,\nu},\ldots,\langle\!\langle t_n \rangle\!\rangle_{M,\theta,\nu}) \end{array}$$

Then, we define inductively the satisfaction relation $\models_{\theta,\nu}$ between colored markings M and CML formulas as follows:

$$\begin{array}{rcl} M \models_{\theta,\nu} \mathit{true} & & \text{always} \\ M \models_{\theta,\nu} x = y & \text{iff} & \theta(x) = \theta(y) \\ M \models_{\theta,\nu} p(x) & \text{iff} & \mathit{place}_M(\theta(x)) = p \\ M \models_{\theta,\nu} r(t_1,\ldots,t_m) & \text{iff} & r(\langle\!\langle t_1 \rangle\!\rangle_{M,\theta,\nu},\ldots,\langle\!\langle t_m \rangle\!\rangle_{M,\theta,\nu}) \\ M \models_{\theta,\nu} \neg\varphi & \text{iff} & M \not\models_{\theta,\nu} \varphi \\ M \models_{\theta,\nu} \varphi_1 \vee \varphi_2 & \text{iff} & M \models_{\theta,\nu} \varphi_1 \text{ or } M \models_{\theta,\nu} \varphi_2 \\ M \models_{\theta,\nu} \exists x.\, \varphi & \text{iff} & \exists t \in \mathbb{N}.\ M \models_{\theta[x \leftarrow t],\nu} \varphi \\ M \models_{\theta,\nu} \exists z.\, \varphi & \text{iff} & \exists c \in \mathbb{C}.\ M \models_{\theta,\nu[z \leftarrow c]} \varphi \end{array}$$

For every formula φ, we define $[\![\varphi]\!]_{\theta,\nu}$ to be the set of colored markings M such that $M \models_{\theta,\nu} \varphi$. A formula φ is *satisfiable* iff there exist valuations θ and ν s.t. $[\![\varphi]\!]_{\theta,\nu} \neq \emptyset$. The subscripts of $\models$ and $[\![\cdot]\!]$ are omitted in the case of a closed formula.

2.3. **Syntactical forms and fragments.**

2.3.1. *Prenex normal form:* A formula is in *prenex normal form* (PNF) if it is of the form

$$Q_1 y_1 Q_2 y_2 \ldots Q_m y_m.\ \varphi$$

where (1) $Q_1,\ldots,Q_m$ are (existential or universal) quantifiers, (2) $y_1,\ldots,y_m$ are variables in $T \cup C$, and φ is a quantifier-free formula. It can be proved that for every formula φ in CML, there exists an equivalent formula φ' in prenex normal form.

2.3.2. *Quantifier alternation hierarchy:* We consider two families $\{\Sigma_n\}_{n\geq 0}$ and $\{\Pi_n\}_{n\geq 0}$ of fragments of CML defined according to the alternation depth of existential and universal quantifiers in their PNF:

- Let $\Sigma_0 = \Pi_0$ be the set of formulas in PNF where all quantified variables are in C,
- For $n \geq 0$, let Σ_{n+1} (resp. Π_{n+1}) be the set of formulas $Qy_1 \ldots y_m.\ \varphi$ in PNF where $y_1,\ldots,y_m \in T \cup C$, Q is the existential (resp. universal) quantifier $\exists$ (resp. $\forall$), and φ is a formula in Π_n (resp. Σ_n).

It is easy to see that, for every $n \geq 0$, Σ_n and Π_n are closed under conjunction and disjunction, and that the negation of a Σ_n formula is a Π_n formula and vice versa. For every $n \geq 0$, let $B(\Sigma_n)$ denote the set of all boolean combinations of Σ_n formulas. Clearly, $B(\Sigma_n)$ subsumes both Σ_n and Π_n, and is included in both Σ_{n+1} and Π_{n+1}.

2.3.3. *Special form:* The set of formulas in special form is given by the grammar:

$$\varphi ::= \mathit{true} \mid x = y \mid r(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists z.\ \varphi \mid \exists x \in p.\ \varphi$$

where $x, y \in T$, $z \in C$, $p \in \mathbb{P} \cup \{\bot\}$, $r \in \Xi$, and $t_1, \ldots, t_n$ are token color terms. So, formulas in special form do not contain atoms of the form $p(x)$.

It is not difficult to see that for every closed formula φ in CML, there exists an equivalent formula φ' in special form. The transformation is based on the following fact: since variables are assumed to be quantified at most once in formulas, each formula $\exists x.\ \phi$ can be replaced by $\bigvee_{p \in \mathbb{P} \cup \{\bot\}} \exists x \in p.\ \phi_{x,p}$ where $\phi_{x,p}$ is obtained by substituting in ϕ each occurrence of $p(x)$ by *true*, and each occurrence of $q(x)$, with $p \neq q$, by *false*.

2.3.4. *Examples of properties expressible in* CML*:* The fact that "the place p is empty" is expressed by the Π_1 formula $\forall x.\ \neg p(x)$. The fact that "p contains precisely one token" is expressed by the $B(\Sigma_1)$ formula: $(\exists x \in p.\ \mathit{true}) \wedge (\forall y, z \in p.\ y = z)$. The Π_1 formula $\forall x, y \in p.\ x = y$ expresses the fact that p has one or zero token.

The properties above do not depend on the colors of the token. The following examples show that the number of tokens in a place is also determined by properties of colors attached to tokens. Let consider now the logic $\mathsf{CML}(\mathbb{N}, \{0\}, \{\leq\})$. Then, the fact that "$p$ contains an infinite number of tokens" is implied by the Π_2 formula:

$$\forall x \in p.\ \exists y \in p.\ \delta_1(x) < \delta_1(y)$$

Conversely, the fact that "p has a finite number of tokens" is implied by the Σ_2 formula:

$$\exists x, y \in p.\ \forall z, u \in p.\ \delta_1(x) \leq \delta_1(z) \leq \delta_1(y) \wedge (\delta_1(z) = \delta_1(u) \Longrightarrow z = u)$$

3. Satisfiability Problem: Undecidability

We show hereafter that the satisfiability problem of the logic CML is undecidable as soon as we consider formulas in Π_2, and this holds even for simple theories on colors.

Theorem 3.1. *The satisfiability problem of the fragment* Π_2 *of* $\mathsf{CML}(\mathbb{N}^2, \{0\}, \{\leq\})$ *is undecidable.*

Proof. The proof is done by reduction of the halting problem of Turing machines. The idea is to encode a computation of a machine, seen as a sequence of tape configurations, using tokens with integer colors. Each token represents a cell in the tape of the machine at some computation step. Therefore, the token has two integer colors: its position in the tape, and the position of its configuration in the computation (the computation step). The place of a token identifies uniquely the letter stored in the associated cell, the control state of the machine in the computation step of the cell, and the position of the head. Then, it is possible to express using formulas in Π_2 that two consecutive configurations correspond indeed to a valid transition of the machine. Intuitively, this is possible because Π_2 formulas allow to relate each cell at some configuration to the corresponding cell at the next configuration.

Let us fix the notations used for Turing machine. A Turing machine is defined by $M = (Q, \Gamma, B, q_0, q_f, \Delta)$ where Q is its finite set of states, Γ is the finite tape alphabet containing the default blank symbol B, $q_0, q_f \in Q$ are the initial resp. the final state, and Δ, called the transition relation, is a subset of $Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$.

A configuration of the machine is given by a triplet $(q, \mathcal{T}, i)$ where $q \in Q$, $\mathcal{T} \in [\mathbb{N} \mapsto \Gamma]$ is the tape of cells identified by their position $j \in \mathbb{N}$ and storing a letter $\mathcal{T}(j) \in \Gamma$, and i is the position of the head on the tape.

A transition $(q, X, q', Y, d) \in \Delta$ defines a relation between two configurations $(q, \mathcal{T}, i)$ and $(q', \mathcal{T}', i')$ iff either $i' = i+1$ and $d = R$ or $i' = i-1$ and $d = L$, the machine reads X at position i, i.e. $\mathcal{T}(i) = X$, and writes Y at the same position, i.e. $\mathcal{T}'(i) = Y$, and in any other position k different from i, the tapes $\mathcal{T}$ and $\mathcal{T}'$ are equal, i.e. $\forall k.\ k \neq i \implies \mathcal{T}(k) = \mathcal{T}'(k)$. The initial configuration of the machine is $(q_0, \mathcal{T}_0, 0)$ where $\mathcal{T}_0$ is the tape with all cells containing the blank symbol B.

Without loss of generality, we suppose that (a) the machine has no deadlocks, (b) the head never goes left when it is at position 0, and (c) when the final state is reached the machine loops in this state.

We proceed now to the encoding of a computation that reaches the final state using a Π_2 formula of $\mathsf{CML}(\mathbb{N}^2, \{0\}, \{\leq\})$.

Instead of generic names δ_1 and δ_2 for color functions we use more intuitive names *step* and *cell* respectively. A token x with $\mathit{step}(x) = j$ and $\mathit{cell}(x) = i$ represents the i^{th} cell of the j^{th} configuration in a computation.

We define the set of places $\mathbb{P} = \Gamma \times \{\mathit{Head}, \mathit{Nohead}\} \times Q$ and, for convenience, we denote members of $\mathbb{P}$ by strings, e.g., $\mathit{A\_Head\_q}$ with $A \in \Gamma$ and $q \in Q$. A token x in a place named $\mathit{A\_Head\_q}$ encodes a cell labeled by the letter A in a configuration where the head is at the position $\mathit{cell}(x)$ and the current state is q. Since in a given configuration the head and the control state have a unique occurence, our encoding includes the property that, among all tokens that have the same *step* color, there is only one token in a place containing Head in its name.

First, we encode the properties of tapes. For this, we introduce the shorthand notation $\mathtt{Head}(x)$, parametrized by a token variable x, expressing that the token represented by x encodes a cell that carries the head, i.e, the name of its place has Head as substring.

$$\mathtt{Head}(x) = \bigvee_{q \in Q} \bigvee_{A \in \Gamma} \mathit{A\_Head\_q}(x)$$

The following Π_2 formula $\mathtt{Tapes}$ expresses that, for any tape j in an infinite computation, any cell i is represented by a unique token x (conditions (3.1) and (3.2)), and there is exactly one token z which represents the position of the head (conditions (3.3) and (3.4)).

$$\begin{aligned}
\mathtt{Tapes} \ &= \ \forall i, j.\ \exists x.\ \mathit{cell}(x) = i \wedge \mathit{step}(x) = j && (3.1)\\
&\wedge \ \forall x, y.\ (\mathit{step}(x) = \mathit{step}(y) \wedge \mathit{cell}(x) = \mathit{cell}(y)) \implies x = y && (3.2)\\
&\wedge \ \forall j.\ \exists x.\ \mathit{step}(x) = j \wedge \mathtt{Head}(x) && (3.3)\\
&\wedge \ \forall x, y.\ (\mathtt{Head}(x) \wedge \mathtt{Head}(y)) \implies (\mathit{step}(x) \neq \mathit{step}(y)) && (3.4)
\end{aligned}$$

Second, we encode the initial configuration using the following $B(\Sigma_1)$ formula:

$$\begin{aligned}
\mathtt{Init} \ &= \ \forall x.\ (\mathit{step}(x) = 0 \wedge \mathit{cell}(x) > 0) \implies \mathit{B\_NotHead\_q_0}(x)\\
&\wedge \ \exists x.\ \mathit{step}(x) = 0 \wedge \mathit{cell}(x) = 0 \wedge \mathit{B\_Head\_q_0}(x)
\end{aligned}$$

Third, we encode the termination condition saying that, at some step, the computation reaches the final state:

$$\mathtt{Acceptance} = \exists x.\ \bigvee_{A \in \Gamma} \mathit{A\_Head\_q_f}(x)$$

Finally, we encode each transition, i.e., the condition defining when two successive configurations correspond to a valid transition in the machine. For this, we have to fix the token storing the head in the current configuration (x), the tokens at the left (x_l) and at the right (x_r) of the head in the current configuration, and the tokens in the next configuration having the same position than x, x_l, and x_r (x', x'_l, resp. x'_r). When this identification is done (see the left part of the implication), we have to decompose the global transition over all transitions $\delta \in \Delta$:

$$\begin{array}{rl} \texttt{Trans} = & \forall x, x_l, x_r.\ \forall x', x'_l, x'_r. \\ & \left(\begin{array}{cl} & \texttt{Head}(x) \\ \wedge & \mathit{step}(x) = \mathit{step}(x_l) \wedge \mathit{step}(x) = \mathit{step}(x_r) \\ \wedge & \neg(\exists y.\ \mathit{cell}(x_l) < \mathit{cell}(y) < \mathit{cell}(x)) \\ \wedge & \neg(\exists y.\ \mathit{cell}(x) < \mathit{cell}(y) < \mathit{cell}(x_r)) \\ \wedge & \neg(\exists y.\ \mathit{step}(x) < \mathit{step}(y) < \mathit{step}(x')) \\ \wedge & \mathit{step}(x') = \mathit{step}(x'_l) \wedge \mathit{step}(x') = \mathit{step}(x'_r) \\ \wedge & \mathit{cell}(x) = \mathit{cell}(x') \wedge \mathit{cell}(x_l) = \mathit{cell}(x'_l) \wedge \mathit{cell}(x_r) = \mathit{cell}(x'_r) \end{array} \right) \\ & \quad \Longrightarrow \bigvee_{\delta \in \Delta} \texttt{Trans}_\delta(x, x_l, x_r, x', x'_l, x'_r) \end{array}$$

where $\texttt{Trans}_\delta$ relates its parameters accordingly to transition δ. For example, if the transition δ is of the form (q, X, q', Y, L) (the case of head moving at right is symmetrical), then we obtain the following Π_1 formula:

$$\begin{array}{rcl} \texttt{Trans}_\delta(x, x_l, x_r, x', x'_l, x'_r) = & & X\_Head\_q(x) \wedge Y\_NotHead\_q'(x') \\ & \wedge & \bigwedge_{A \in \Gamma}(A\_NotHead\_q(x_l) \Longrightarrow A\_Head\_q'(x'_l)) \\ & \wedge & \forall y, y'. \left(\begin{array}{cl} & y \neq x \wedge y \neq x_l \\ \wedge & y' \neq x' \wedge y' \neq x'_l \\ \wedge & \mathit{step}(y) = \mathit{step}(x) \\ \wedge & \mathit{step}(y') = \mathit{step}(x') \\ \wedge & \mathit{cell}(y) = \mathit{cell}(y') \end{array} \right) \Longrightarrow \texttt{Same}(y, y') \end{array}$$

where the shorthand notation $\texttt{Same}(y, y')$ stands for

$$\bigwedge_{A \in \Gamma, p \in Q} A\_NotHead\_p(y) \Leftrightarrow A\_NotHead\_p(y')$$

and expresses that the two tokens y and y' carry the same letter. Then, the *Trans* formula is in $B(\Sigma_1)$.

The conjunction $\texttt{Tapes} \wedge \texttt{Init} \wedge \texttt{Trans} \wedge \texttt{Acceptance}$ is a Π_2 formula which is satisfiable iff there is an accepting run. This reduction shows the undecidability of satisfiability for Π_2 fragment of $\mathsf{CML}(\mathbb{N}^2, \{0\}, \{\leq\})$. □

4. Satisfiability problem: A Generic Decidability Result

We prove in this section that the satisfiability problem for formulas in the fragment Σ_2 of CML is decidable whenever this problem is decidable for the underlying color logic.

Theorem 4.1. *The satisfiability problem of the fragment Σ_2 of $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$, for any $N \geq 1$, is decidable provided that the satisfiability problem of $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$ is decidable.*

Proof. The idea of the proof is to reduce the satisfiability problem of Σ_2 formulas to the satisfiability problem of Σ_0 formulas. We proceed as follows: we prove first that the fragment Σ_2 has the small model property, i.e., every satisfiable formula φ in Σ_2 has a model of a bounded size (where the size is the number of tokens in each place). This bound corresponds actually to the number of existentially quantified token variables in the formula. Notice that this fact does not lead directly to an enumerative decision procedure for the satisfiability problem since the number of models of a bounded size is infinite in general (due to infinite color domains). Then we use the fact that over a finite model, the universal quantifications in φ can be transformed into finite conjunctions in order to build a formula $\widehat{\varphi}$ in Σ_1 which is satisfiable if and only if the original formula φ is satisfiable. Actually, $\widehat{\varphi}$ defines precisely the upward-closure of the set of markings defined by φ (w.r.t. the inclusion ordering between sets of colored markings, extended to vectors of places). Finally we show that the Σ_1 formula $\widehat{\varphi}$ is satisfiable if and only if the Σ_0 formula obtained by transforming existential quantification over tokens into existential quantification over colors is decidable.

We define the size of a marking M to be the number of tokens x for which $place_M(x) \neq \perp$. A marking M' is said to be a sub-marking of a marking M if all tokens in M' for which $place_M(x) \neq \perp$ are mapped identically by M and M'. We also define the upward closure of a set of markings $\mathcal{M}$ to be the set of all the markings that have a sub-marking in $\mathcal{M}$.

First, we show the following lemma:

Lemma 4.2. *Let φ be a Σ_2 closed formula $\varphi = \exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \forall \overrightarrow{y}.\ \phi$ where $\overrightarrow{x}$ and $\overrightarrow{y}$ are token variables, $\overrightarrow{z}$ are color variables, and ϕ is a Σ_0 formula. Then:*

(1) *φ has a model iff it has a model of size less than or equal to $|\overrightarrow{x}|$.*
(2) *The upward closure of $[\![\varphi]\!]$ w.r.t. the sub-marking ordering is effectively definable in Σ_1.*

Proof. Point (1): ($\Leftarrow$) Immediate.
($\Rightarrow$) Let M be a model of φ. Then, there exists a vector of tokens $\overrightarrow{t} \subset \mathbb{N}$, a vector of colors $\overrightarrow{c} \subset \mathbb{C}$, and two mappings $\theta : \overrightarrow{x} \mapsto \overrightarrow{t}$ and $\nu : \overrightarrow{z} \mapsto \overrightarrow{c}$ such that $M \models_{\theta,\nu} \forall \overrightarrow{y}.\ \phi$.

Given any universally quantified formula it is always the case that if it is satisfied by a marking then it is also satisfied by all its sub-markings (w.r.t inclusion ordering). In particular, we define M' to be the sub-marking of M that agrees only on tokens in $\overrightarrow{t}$. Then, we have $M' \models_{\theta,\nu} \forall \overrightarrow{y}.\ \phi$, and therefore $M' \models \exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \forall \overrightarrow{y}.\ \phi$. Therefore, for the fragment Σ_2, every satisfiable formula $\varphi = \exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \forall \overrightarrow{y}.\ \phi$ has a model of size less or equal than $|\overrightarrow{x}|$. However this fact does not imply the decidability of the satisfiability problem since the color domain is infinite.

Point (2): We show that for any formula φ in Σ_2 it exists a formula $\widehat{\varphi}$ such that any model M of φ has a sub-marking M' which is a model of $\widehat{\varphi}$, i.e., the upper closure of the set of models of φ is given by the set of models of $\widehat{\varphi}$.

Let Θ be the set of all (partial or total) mappings σ from elements of $\overrightarrow{y}$ to elements of $\overrightarrow{x}$. Then, we have that any model M of φ is also a model of $\exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \varphi^{(1)}$ where

$$\varphi^{(1)} = \bigwedge_{\sigma \in \Theta} \forall \overrightarrow{y}.\ \Big(\big((\bigwedge_{y \in dom(\sigma)} y = \sigma(y)) \wedge (\bigwedge_{y \notin dom(\sigma)} \bigwedge_{x \in \overrightarrow{x}} y \neq x)\big) \Longrightarrow \varphi\Big)$$

This means that there exists a vector of tokens $\overrightarrow{t} \subset \mathbb{N}$, a vector of colors $\overrightarrow{c} \subset \mathbb{C}$, and two mappings $\theta : \overrightarrow{x} \mapsto \overrightarrow{t}$ and $\nu : \overrightarrow{z} \mapsto \overrightarrow{c}$ such that $M \models_{\theta,\nu} \varphi^{(1)}$. Consider now M' to be the sub-marking of M that agrees only on tokens in $\overrightarrow{t}$. Then, $M \models_{\theta,\nu} \varphi^{(1)}$ implies that:

$$M' \models_{\theta,\nu} \bigwedge_{\substack{\sigma\in\Theta\\ dom(\sigma)=\overrightarrow{y}}} \forall \overrightarrow{y}.\ \big((\bigwedge_{y\in\overrightarrow{y}} y = \sigma(y)) \Longrightarrow \varphi\big)$$

which is equivalent to $M' \models \widehat{\varphi}$ with:

$$\widehat{\varphi} = \exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \bigwedge_{\substack{\sigma\in\Theta\\ dom(\sigma)=\overrightarrow{y}}} \varphi[\sigma(\overrightarrow{y})/\overrightarrow{y}]$$

By definition of $\widehat{\varphi}$, any of its minimal models is also a model of φ, and any of the models of φ has a sub-model that is a model of $\widehat{\varphi}$. □

A direct consequence of the lemma above is that it is possible to reduce the satisfiability problem from Σ_2 to Σ_1. To prove the main theorem, we have to show that the satisfiability problem of Σ_1 can be reduced to one of Σ_0. Let us consider a Σ_1 formula $\varphi = \exists \overrightarrow{x}.\ \phi$ with ϕ in Σ_0.

We do the following transformations: (1) we eliminate token equality by enumerating all the possible equivalence classes for equality between the finite number of variables in $\overrightarrow{x}$, then (2) we eliminate formulas of the form $p(x)$ by enumerating all the possible mappings from a token variable x to places in $\mathbb{P}$, and (3) we replace terms of the form $\delta_k(x)$ by fresh color variables. Let us describe more formally these three transformations.

Step 1: Let $\mathcal{B}(\overrightarrow{x})$ be the set of all possible equivalence classes (w.r.t. the equality relation) over elements of $\overrightarrow{x}$: an element e in $\mathcal{B}(\overrightarrow{x})$ is a mapping from $\overrightarrow{x}$ to a vector of variables $\overrightarrow{x}^{(e)} \subseteq \overrightarrow{x}$ that contains only one variable for each equivalence class.

We define ϕ_e to be $\phi[\overrightarrow{x}^{(e)}/\overrightarrow{x}]$ where, after the substitution, each atomic formula that is a token equality is replaced by "*true*" if it is a trivial equality $x = x$ and by "*false*" otherwise. Clearly φ is equivalent to

$$\bigvee_{e\in\mathcal{B}(\overrightarrow{x})} \exists \overrightarrow{x}^{(e)}.\ \bigwedge_{i\neq j}(x_i^{(e)} \neq x_j^{(e)}) \wedge \phi_e$$

Step 2: Similarly, we eliminate from ϕ_e the occurrences of formulas $p(x)$. For a mapping $\sigma \in [\overrightarrow{x}^{(e)} \rightarrow \mathbb{P}]$ and a variable x, $\sigma(x)(x)$ is a formula saying that the variable x is in the place $\sigma(x)$. We use the notation $\sigma(\overrightarrow{x})(\overrightarrow{x})$ instead of $\bigwedge_i \sigma(x_i)(x_i)$. Again, for each value of σ and e we define $\phi_{e,\sigma}$ to be ϕ_e where each atomic sub-formula $p(x)$ is replaced by "*true*" if $\sigma(x) = p$ and by "*false*" otherwise.

Then, we obtain an equivalent formula $\varphi_{=,p}$:

$$\bigvee_{e\in\mathcal{B}(\overrightarrow{x})} \exists \overrightarrow{x}^{(e)}.\ \bigwedge_{i\neq j}(x_i^{(e)} \neq x_j^{(e)}) \wedge \bigvee_{\sigma\in[\overrightarrow{x}^{(e)}\rightarrow\mathbb{P}]} \sigma(\overrightarrow{x}^{(e)})(\overrightarrow{x}^{(e)}) \wedge \phi_{e,\sigma}$$

where sub-formulas $\phi_{e,\sigma}$ do not contain any atoms of the form $x_i^{(e)} = x_j^{(e)}$ or $p(x_i^{(e)})$. Still, $\varphi_{e,\sigma}$ is not a Σ_0 formula, because it contains terms of the form $\delta_k(x)$.

Step 3: For each coloring symbol δ_k and each token variable $x \in \overrightarrow{x}^{(e)}$, we define a color variable $s_{k,x}$. Let $\overrightarrow{s}^{(e)}$ be a vector containing all such color variables for each variable in $\overrightarrow{x}^{(e)}$. Then the formula $\varphi_{=,p}$ is satisfiable iff the following Σ_0 formula is satisfiable:

$$\bigvee_{e\in\mathcal{B}(\overrightarrow{x})} \exists \overrightarrow{s_e}. \bigvee_{\sigma\in[\overrightarrow{x}^{(e)}\to\mathbb{P}]} \phi_{e,\sigma}[s_{k,x}/\delta_k(x)]_{1\leq k\leq N, x\in\overrightarrow{x}^{(e)}}$$

Therefore, the satisfiability problem of Σ_2 can be reduced to satisfiability problem of Σ_0, which is decidable by hypothesis. □

Complexity: From the last part of the proof, it follows that the satisfiability problem of a Σ_1 formula can be reduced in NP time to the satisfiability problem of a formula in the color logic $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$. Indeed, in *Step 1* an equivalence relation between the existentially quantified variables $\overrightarrow{x}$ is guessed and in *Step 2* a place in $\mathbb{P}$ for the representative of each equivalence class is guessed, and given these guesses, a Σ_0 formula of linear size (w.r.t. the size of the original Σ_1) is built.

From the first part of the proof, it follows that the reduction from the satisfiability problem of a Σ_2 formula to the satisfiability of a Σ_1 formula is in general exponential. More precisely, if $\varphi = \exists\overrightarrow{x}.\ \exists\overrightarrow{z}.\ \forall\overrightarrow{y}.\ \phi$ is a Σ_2 formula, then the equi-satisfiable Σ_1 formula $\widehat{\varphi}$ is of size $O(|\overrightarrow{x}|^{|\overrightarrow{y}|}|\varphi|)$. Therefore, the reduction of a Σ_2 formula to an equi-satisfiable formula in Σ_0 is in NEXPTIME.

If the number of universally quantified variables (i.e., $|\overrightarrow{y}|$) is fixed, the reduction to an equi-satisfiable Σ_1 formula $\widehat{\varphi}$ becomes polynomial in the number of existentially quantified variables (i.e., $|\overrightarrow{x}|$). Then, in this case, the complexity of the reduction from Σ_2 formulas to equi-satisfiable Σ_0 formulas is in NP.

5. Constrained Petri Nets

We introduce hereafter models for networks of processes based on multiset rewriting systems with data.

A *Constrained Petri Net* (CPN) over the logic $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$ is a tuple $S = (\mathbb{P}, \Delta)$ where $\mathbb{P}$ is a finite set of places used in CML, and Δ is a finite set of *constrained transitions* of the form:

$$p_1, \ldots, p_n \ \hookrightarrow\ q_1, \ldots, q_m \ :\ \varphi \tag{5.1}$$

where $p_i, q_j \in \mathbb{P}$ for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, and φ is a $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$ formula called the *transition guard* such that (1) $FV(\varphi) = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\}$, and (2) all occurences of variables y_j in φ, for any $j \in \{1, \ldots, m\}$, are in terms of the form $\delta_k(y_j)$, for some $k \in \{1, \ldots, N\}$.

Configurations of CPNs are colored markings. Intuitively, the application of a constrained transition to a colored marking M (leading to a colored marking M') consists in (1) deleting tokens represented by the variables x_i from the corresponding places p_i, and in (2) creating tokens represented by variables y_j in the places q_j, provided that the formula φ is satisfied. The formula φ expresses constraints on the tokens in the marking M (especially on the tokens which are deleted) as well as constraints on the colors of created tokens (relating these colors with those of the tokens in M).

Formally, given a CPN S, we define a transition relation $\rightarrow_S$ between colored markings as follows: for every two colored markings M and M', we have $M \rightarrow_S M'$ iff there exists a constrained transition of the form (5.1), and there exist tokens $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ s.t. $\forall i, j \in \{1, \ldots, n\}.\ i \neq j \Rightarrow t_i \neq t_j$, and $\forall i, j \in \{1, \ldots, m\}.\ i \neq j \Rightarrow t'_i \neq t'_j$, and

(1) $\forall i \in \{1, \ldots, n\}.\ \mathit{place}_M(t_i) = p_i$ and $\mathit{place}_{M'}(t_i) = \bot$,
(2) $\forall i \in \{1, \ldots, m\}.\ \mathit{place}_M(t'_i) = \bot$ and $\mathit{place}_{M'}(t'_i) = q_i$,
(3) $\forall t \in \mathbb{N}$, if $\forall i \in \{1, \ldots, n\}.\ t \neq t_i$ and $\forall j \in \{1, \ldots, m\}.\ t \neq t'_j$, then $M(t) = M'(t)$,
(4) $M \models_{\theta, \nu_\emptyset} \varphi[\mathit{color}_{M',k}(t'_j)/\delta_k(y_j)]_{1 \leq k \leq N, 1 \leq j \leq m}$, where $\theta \in [T \rightarrow \mathbb{N}]$ is a valuation of token variables such that $\forall i \in \{1, \ldots, n\}.\ \theta(x_i) = t_i$, and $\nu_\emptyset$ is the empty domain valuation of color variables.

Given a colored marking M let $\mathsf{post}_S(M) = \{M' : M \rightarrow_S M'\}$ be the set of all immediate successors of M, and let $\mathsf{pre}_S(M) = \{M' : M' \rightarrow_S M\}$ be the set of all immediate predecessors of M. These definitions can be generalized straightforwardly to sets of markings. Given a set of colored markings $\mathcal{M}$, let $\widetilde{pre}_S(\mathcal{M}) = \overline{\mathsf{pre}_S(\overline{\mathcal{M}})}$, where $(\overline{\cdot})$ denotes complementation (w.r.t. the set of all colored markings).

Given a fragment Θ of CML, we denote by $\mathsf{CPN}[\Theta]$ the class of CPN where all transition guards are formulas in the fragment Θ. Due to the (un)decidability results of sections 3 and 4, we focus in the sequel on the classes $\mathsf{CPN}[\Sigma_2]$ and $\mathsf{CPN}[\Sigma_1]$.

6. Modeling Power of CPN

We show in this section how constrained Petri nets can be used to model (unbounded) dynamic networks of parallel processes. We assume that each process is defined by an extended automaton, i.e., a finite-control state machine supplied with variables and data structures ranging over potentially infinite domains (such as integer variables, reals, etc). Processes running in parallel can communicate and synchronize using various kinds of mechanisms (rendez-vous, shared variables, locks, etc). Moreover, they can dynamically spawn new (copies of) processes in the network.

More precisely, let $\mathcal{Q}$ be the finite set of control locations of the extended automata, and let $\overrightarrow{l} = (l_1, \ldots, l_N)$ and $\overrightarrow{g} = (g_1, \ldots, g_G)$ be the sets of local respectively global variables manipulated by these automata. Transitions between control locations are labeled by actions which combine (1) tests over the values of local/global variables, (2) assignments of local/global variables, (3) creation of a new process in a control location, (4) synchronization (e.g., CCS-like rendez-vous, locks, priorities, etc.). Tests over variables are first-order assertions based on a set of predicates Ξ. Variables are assigned with expressions built from local and global variables using a set of operations Ω.

Example 6.1. Reader-writer is a classical synchronization scheme used in operating systems or other large scale systems. It allows processes to work (read and write) on shared data. Reader processes may read data in parallel but they are exclusive with writers. Writer processes can only work in exclusive mode with other processes. A *reader-writer lock* is used to implement such kind of synchronization for any number of readers and writers. For this, readers have to acquire the lock in *read mode* and writers in *write mode.*

Let us consider the program proposed in [FFQ02] and using the reader-writer lock given in Table 6.1. It consists of several Reader and Writer processes. The code of each process is given in Table 6.1. (To keep the example readable, we omit the processes spawning the readers and writers.) The program uses a global reader-writer lock variable l and a global variable x representing the shared data. Each Reader process has a local variable y.

```
    process Writer:                  process Reader:
1:   l.acq_write(_pid);         1:   l.acq_read(_pid);
2:   x = g(x);                  2:   y = f(x);
3:   l.rel_write(_pid);         3:   l.rel_read(_pid);
4:                              4:
```

Table 1: Example of program using reader-writer lock.

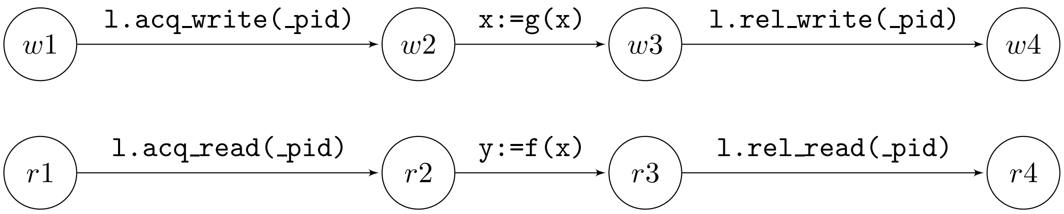

Figure 1: Extended automata model for the program in Table 6.1.

Moreover, each process has a unique identifier represented by the _pid local variable. Let us assume that x, y, and _pid are of integer type. Writer processes change the value of the global variable x after acquiring the lock in write mode. Reader processes are setting their local variable y to a value depending on x after acquiring the lock in read mode.

Then, the extended automata model for the program in Table 6.1 is obtained by associating a control location to each line of the program and by labeling transitions between control locations with the statements of the program. The extended automata model is provided on Figure 6.1.

We show hereafter how to build a CPN model for a network of extended automata described above. The logic of markings used by the CPN model is defined by $\mathsf{CML}(\mathbb{C}^N, \Omega, \Xi)$ where $N \geq 1$ is the (maximal) number of local variables of each process. To each control location in $\mathcal{Q}$ and to each global variable in $\overrightarrow{g}$ is associated a unique place in $\mathbb{P}$. Then, each running process is represented by a token, and in every marking, the place associated with the control location $q \in \mathcal{Q}$ contains precisely the tokens representing processes which are at the control location q. The value of a local variable l_i of a process represented by token t is given by $\delta_i(t)$. For global variables which are scalar, the place associated in $\mathbb{P}$ (for convenience, we use the same name for the place and the global variable) contains a single token whose first color stores the current value of the global variable. Global variables representing parametric-size collections may also be modeled by a place storing for each element of the collection a token whose first color gives the value of the element. However, we cannot express in the decidable fragment Σ_2 of CML the fact that a multiset indeed encodes an array of elements indexed by integers in some given interval. The reason is that, while we can express in Π_1 the fact that each token has a unique color in the interval, we need to use Π_2 formulas to say that for each color in the interval there exists a token with that color. Nevertheless, for the verification of safety properties and for checking invariants, it is not necessary to require the latter property.

The set of constrained transitions of the CPN associated with the network are obtained using the following general rules:

Test: A process action $q \xrightarrow{\varphi(\overrightarrow{l},\overrightarrow{g})} q'$ where φ is a $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$ formula, is modeled by:

$$q, g_1, \ldots, g_G \;\hookrightarrow\; q', g_1, \ldots, g_G \;:\; \varphi\eta \wedge \bigwedge_{i=1}^{G+1} \varphi_{id}(i)$$

where η is the substitution $[\delta_k(x_1)/l_k]_{1 \leq k \leq N}[\delta_1(x_{k+1})/g_k]_{1 \leq k \leq G}$, and

$$\varphi_{id}(i) = \bigwedge_{j=1}^{N} \delta_j(y_i) = \delta_j(x_i)$$

Assignment: A process action $q \xrightarrow{(\overrightarrow{l},\overrightarrow{g}) := \overrightarrow{t}(\overrightarrow{l},\overrightarrow{g})} q'$ where $\overrightarrow{t}$ is a vector of $N + G$ Ω-terms, is modeled by:

$$q, g_1, \ldots, g_G \;\hookrightarrow\; q', g_1, \ldots, g_G \;:\; \bigwedge_{i=1}^{N} \delta_i(y_1) = t_i\eta \;\wedge\; \bigwedge_{j=1}^{G} \delta_1(y_{j+1}) = t_{N+j}\eta$$

where η is the substitution defined in the previous case.

In the modeling above, we consider that the execution of the process action is atomic. When tests and assignments are not atomic, we must transform each of them into a sequence of atomic operations: read first the global variables and assign their values to local variables, compute locally the new values to be assigned/tested, and finally, assign/test these values.

Process creation: An action spawning a new process $q \xrightarrow{\mathsf{spawn}(q_0)} q'$ is modeled using a transition which creates a new token in the initial control location q_0 of the new process:

$$q \hookrightarrow q', q_0 \;:\; \varphi_{id}(1) \wedge \varphi_0$$

where φ_0 is $\bigwedge_{i=1}^{N} \delta_i(y_2) = \mathit{null}$ with *null* the general initial value for local variables.

Moreover, it is possible to associate with each newly created process an identity classically defined by a positive integer number. For that, let us consider that the first color δ_1 gives the identity of the process represented by the token. To ensure that different processes have different identities, we express in the guard of every transition which creates a process the fact that the identity of this process does not exist already among tokens in places corresponding to control locations. This can easily be done using a universally quantified (Π_1) formula. Therefore, a spawn action $q \xrightarrow{\mathsf{spawn}(q_0)} q'$ is modeled by:

$$q \hookrightarrow q', q_0 \;:\; \varphi_{id}(1) \wedge \varphi_0'$$

where

$$\varphi_0' = \bigwedge_{i=2}^{N} \delta_i(y_2) = \mathit{null} \;\wedge\; \bigwedge_{\ell \in \mathcal{Q}} \forall t \in \ell.\ \neg(\delta_1(y_2) = \delta_1(t))$$

The modeling of other actions (such as local/global variables assignment/test) can be modified accordingly in order to propagate the process identity through the transition. Notice that process identities are different from token values. Indeed, in some cases (e.g., for modeling value passing as described further in this section), we may use different tokens (at some special places representing buffers for instance) having the same identity δ_1.

$w_1 :$	$w1, w$	$\hookrightarrow$	$w2, w$	:	$\neg(\exists z \in r.\ true) \wedge \delta_2(x_2) < 0 \wedge \delta_2(y_2) = \delta_1(x_1) \wedge$ $\delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1)$
$w_2 :$	$w2, x$	$\hookrightarrow$	$w3, x$	:	$\delta_2(y_2) = g(\delta_2(x_2)) \wedge \delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1)$
$w_3 :$	$w3, w$	$\hookrightarrow$	$w4, w$	:	$\delta_2(x_2) = \delta_1(x_1) \wedge \delta_2(y_2) = -1 \wedge \delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1)$
$r_1 :$	$r1$	$\hookrightarrow$	$r2, r$	:	$(\forall z \in w.\ \delta_2(z) < 0) \wedge \delta_1(y_2) = \delta_1(x_1) \wedge \varphi_{id}(1)$
$r_2 :$	$r2, x$	$\hookrightarrow$	$r3, x$	:	$\delta_2(y_1) = f(\delta_2(x_2)) \ \wedge\ \delta_1(x_1) = \delta_1(y_1) \ \wedge\ \varphi_{id}(2)$
$r_3 :$	$r3, r$	$\hookrightarrow$	$r4$	:	$\delta_1(x_1) = \delta_1(x_2) \wedge \varphi_{id}(1)$

Table 2: CPN model of reader-writer lock.

Synchronization using locks: Locks can be simply modeled using global variables storing the identity of the owner process, or a special value (e.g. -1) if it is free. A process who acquires the lock must check if it is free, and then write his identity:

$$q, lock \hookrightarrow q', lock \ : \ \delta_1(x_2) = -1 \wedge \delta_1(y_2) = \delta_1(x_1) \wedge ...$$

To release the lock, a process assigns -1 to the lock, which can be modeled in a similar way. Other kinds of locks, such as reader-writer locks, can also be modeled in our framework as we show in the following example.

Example 6.2. Let us consider the extended automaton using the reader-writer lock given on Figure 6.1. For each of its states we introduce a place (e.g., place $r3$ for state r3). For the scalar global variable x, we create a place x containing a single token.

The global variable representing the reader-writer lock is modeled following the classical implementation [Ari05] which uses two variables:

- a global *integer* w to store the identifier of the process holding the lock in write mode or -1 if no such process exists (process identifiers are supposed to be positive integers), and
- a global *set of integers* r to represent the processes holding the lock in read mode.

Acquire (acq_read, acq_write) and release (rel_read, rel_write) operations are accessing variables w and r atomically. Then, we introduce a place w (containing a single token) for the scalar global variable w. For the global set variable r we introduce a place which contains a token for each Reader process owning the lock. By consequence, we need two colors for each token in the system: δ_1 to store the identity of processes and δ_2 to store the local variable y for tokens representing Reader processes and the value of global variables w and x for tokens in places w resp. x.

Therefore, the CPN model obtained is defined over the logic $\mathsf{CML}(\mathbb{N}^2, \{0, f, g\}, \{\leq\})$, its set of places is $\mathbb{P} = \{r1, r2, r3, r4, w1, w2, w3, w4, r, w, x\}$, and its transition set Δ is given in Table 6.2. This model belongs to the class $\mathsf{CPN}[\Pi_1]$.

Value passing, return values: Processes may pass/wait for values to/from other processes with specific identities. They can use for that shared arrays of data indexed by process identities. Such an array A can be modeled in our framework using a special place containing for each process a token. Initially, this place is empty, and whenever a new process is created, a token with the same identity is added to this place. Then, to model that a process reads/writes on $A[k]$, we use a transition which takes from the place associated with A the token with color δ_1 equal to i, reads/modifies the value attached with this token, and puts the token again in the same place. For instance, an assignment action $q \xrightarrow{A[k]:=e} q'$ executed by some process is modeled by the transition:

$$q, A \hookrightarrow q', A \;:\; \delta_1(x_2) = k \;\wedge\; \delta_2(y_2) = e \;\wedge\; \delta_1(y_2) = \delta_1(x_2) \;\wedge\; \varphi_{id}(1)$$

Rendez-vous synchronization: Synchronization between a finite number of processes can be modeled as in Petri nets. CPNs allow in addition to put constraints on the colors (data) of the involved processes.

Priorities: Various notion of priorities, such as priorities between different classes of processes (defined by properties of their colors), or priorities between different actions, can be modeled in CPNs. This can be done by imposing in transition guards that transitions (performed by processes or corresponding to actions) of higher priority are not enabled. These constraints can be expressed using Π_1 formulas. In particular, checking that a place p is empty can be expressed by $\forall x.\ \neg p(x)$. (Which shows that, as soon as universally quantified formulas are allowed in guards, our models are as powerful as Turing machines, even for color logics over finite domains.)

7. Computing Post and Pre Images

We address in this section the problem of characterizing in CML the immediate successors/predecessors of CML definable sets of colored markings.

Theorem 7.1. *Let S be a $\mathsf{CPN}[\Sigma_n]$, for $n \in \{1, 2\}$. Then, for every CML closed formula φ in the fragment Σ_n, the sets $\mathsf{post}_S([\![\varphi]\!])$ and $\mathsf{pre}_S([\![\varphi]\!])$ are effectively definable by CML formulas in the same fragment Σ_n.*

Proof. Let φ be a closed formula, and let τ be a transition $\overrightarrow{p} \hookrightarrow \overrightarrow{q} : \psi$ of the system S. W.l.o.g., we suppose that φ and ψ are in special form (see definition in Section 2.3.3). Moreover, we suppose that variables in $\overrightarrow{x}$ and $\overrightarrow{y}$ introduced by τ have fresh names, i.e., different from those of variables quantified in φ and ψ. We define hereafter the formulas $\varphi_{\mathsf{post}} = \mathsf{post}_S([\![\varphi]\!])$ and $\varphi_{\mathsf{pre}} = \mathsf{pre}_S([\![\varphi]\!])$ for this single transition. The generalization to the set of all transitions is straightforward.

The construction of the formulas φ_{post} and φ_{pre} is not trivial because our logic does not allow to use quantification over places and color mappings in $[\mathbb{N} \to \mathbb{C}]$. Intuitively, the idea is to express first the effect of deleting/adding tokens, and then composing these operations to compute the effect of a transition.

Let us introduce two transformations $\ominus$ and $\oplus$ corresponding to deletion and creation of tokens. These operations are inductively defined on the structure of special form formulas in Tables 3 and 4.

The operation $\ominus$ is parameterized by a vector $\overrightarrow{z}$ of token variables to be deleted, a mapping `loc` associating with token variables in $\overrightarrow{z}$ the places from which they will be deleted, and a mapping `col` associating with each token variable in $\overrightarrow{z}$ and eack $k \in \{1, \ldots, N\}$ a fresh color variable in C. Intuitively, $\ominus$ projects a formula on all variables in $\overrightarrow{z}$. Rule $\ominus_2$ substitutes in a color formula $r(\overrightarrow{t})$ all occurences of colored tokens in $\overrightarrow{z}$ by fresh color variables given by the mapping `col`. A formula $x = y$ is unchanged by the application of $\ominus$ if the token variables x and y are not in $\overrightarrow{z}$; otherwise, rule $\ominus_3$ replaces $x = y$ by "*true*" if it is trivially true (i.e., we have the same variable in both sides of the

$$
\begin{array}{lrcl}
\ominus_1 : & \mathit{true} \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \mathit{true} \\
\ominus_2 : & r(\overrightarrow{t}\,) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & r(\overrightarrow{t}\,)[\mathtt{col}(z)(k)/\delta_k(z)]_{1 \leq k \leq N, z \in \overrightarrow{z}} \\
\ominus_3 : & (x = y) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \begin{cases} x = y & \text{if } x, y \notin \overrightarrow{z} \\ \mathit{true} & \text{if } x \equiv y \\ \mathit{false} & \text{otherwise} \end{cases} \\
\ominus_4 : & (\neg\varphi) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \neg(\varphi \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \\
\ominus_5 : & (\varphi_1 \vee \varphi_2) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & (\varphi_1 \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \vee (\varphi_2 \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \\
\ominus_6 : & (\exists x \in p.\ \varphi) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \exists x \in p.\ (\varphi \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \vee \\
& & & \qquad \bigvee_{z \in \overrightarrow{z} : \mathtt{loc}(z) = p} (\varphi[z/x]) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})
\end{array}
$$

Table 3: Definition of the $\ominus$ operator.

$$
\begin{array}{lrcl}
\oplus_1 : & \mathit{true} \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \mathit{true} \\
\oplus_2 : & r(\overrightarrow{t}\,) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & r(\overrightarrow{t}\,) \\
\oplus_3 : & (x = y) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \begin{cases} x = y & \text{if } x, y \notin \overrightarrow{z} \\ \mathit{true} & \text{if } x \equiv y \\ \mathit{false} & \text{otherwise} \end{cases} \\
\oplus_4 : & (\neg\varphi) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \neg(\varphi \oplus (\overrightarrow{z}, \mathtt{loc})) \\
\oplus_5 : & (\varphi_1 \vee \varphi_2) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & (\varphi_1 \oplus (\overrightarrow{z}, \mathtt{loc})) \vee (\varphi_2 \oplus (\overrightarrow{z}, \mathtt{loc})) \\
\oplus_6 : & (\exists x \in p.\ \varphi) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \exists x \in p.\ (\varphi \oplus (\overrightarrow{z}, \mathtt{loc})) \wedge \bigwedge_{z \in \overrightarrow{z} : \mathtt{loc}(z) = p} \neg(x = z)
\end{array}
$$

Table 4: Definition of the $\oplus$ operator.

equality) or by "*false*" if x (or y) is in $\overrightarrow{z}$. Indeed, each token variable in $\overrightarrow{z}$ represents (by the semantics of CPN) a different token, and since this token is deleted by the transition rule, it cannot appear in the reached configuration. Rules $\ominus_4$ and $\ominus_5$ are straightforward. Finally, rule $\ominus_6$ does a case splitting according to the fact whether a deleted token is precisely the one referenced by the existential token quantification or not.

The operation $\oplus$ is parameterized by a vector $\overrightarrow{z}$ of token variables to be added and a mapping $\mathtt{loc}$ associating with each variable in $\overrightarrow{z}$ the place in which it will be added. Intuitively, $\oplus$ transforms a formula taking into account that the tokens added by the transition were not present in the previous configuration (and therefore not constrained by the original formula describing the configuration before the transition). Then, the application of $\oplus$ has no effect on color formulas $r(\overrightarrow{t}\,)$ (rule $\oplus_2$). When equality of tokens is tested, rule $\oplus_3$ takes into account that all added tokens are distinct and different from the existing tokens. For token quantification, rule $\oplus_6$ says that quantified tokens of the previous configuration cannot be equal to the added tokens.

Therefore, we define $\varphi_{\mathsf{post}_\tau}$ to be the formula:

$$\exists \overrightarrow{y} \in \overrightarrow{q}.\ \exists \overrightarrow{c}.\ \big((\varphi \wedge \psi) \ominus (\overrightarrow{x}, \overrightarrow{x} \mapsto \overrightarrow{p}, \overrightarrow{x} \mapsto [1, N] \mapsto \overrightarrow{c})\big)\big) \oplus (\overrightarrow{y}, \overrightarrow{y} \mapsto \overrightarrow{q}) \tag{7.1}$$

In the formula above, we first delete the tokens corresponding to $\overrightarrow{x}$ from the current configuration φ intersected with the guard of the rule ψ. Then, we add tokens corresponding to $\overrightarrow{y}$. Finally, we close the formula by quantifying existentially (1) the color variables $\overrightarrow{c}$ corresponding to colors of deleted tokens $\overrightarrow{x}$ and (2) the token variables $\overrightarrow{y}$ corresponding to the added tokens.

Similarly, we define $\varphi_{\mathsf{pre}_\tau}$ to be the formula:

$$\exists \overrightarrow{x} \in \overrightarrow{p}.\, \exists \overrightarrow{c}.\, \big((\varphi \oplus (\overrightarrow{x}, \overrightarrow{x} \mapsto \overrightarrow{p})) \wedge \psi\big) \ominus (\overrightarrow{y}, \overrightarrow{y} \mapsto \overrightarrow{q}, \overrightarrow{y} \mapsto [1, N] \mapsto \overrightarrow{c})) \qquad (7.2)$$

In the formula above, we first add to the current configuration the tokens represented by the left hand side of the rule $\overrightarrow{x}$ in order to obtain a configuration on which the guard ψ can be applied. Then, we remove the tokens added by the rule using token variables $\overrightarrow{y}$. Finally, we close the formula by quantifying existentially (1) the color variables $\overrightarrow{c}$ corresponding to colors of removed tokens $\overrightarrow{y}$ and (2) the token variables $\overrightarrow{x}$ corresponding to the added tokens. It is easy to see that if φ and ψ are in the Σ_n fragment, for any $n \geq 1$, then both of the formulas $\varphi_{\mathsf{post}_\tau}$ and $\varphi_{\mathsf{pre}_\tau}$ are also in the same fragment Σ_n. □

Complexity: Let φ be a Σ_2 formula, and let $\tau = \overrightarrow{p} \hookrightarrow \overrightarrow{q} : \psi$ be a transition of a system $S \in \mathsf{CPN}[\Sigma_2]$. Then the sizes of formulas $\mathsf{post}_\tau(\varphi)$ and $\mathsf{pre}_\tau(\varphi)$ are in general exponential in the number of quantifiers in $\varphi \wedge \psi$. More precisely, the size of the post (resp. pre) image of φ is $O(|\overrightarrow{p}|^n)$ (resp. $O(|\overrightarrow{q}|^n)$) times greater than the size of the formula $\varphi \wedge \psi$, where n is the number of quantifiers in $\varphi \wedge \psi$. This exponential blow-up is due to the rule $\ominus_6$ in Table 3. If the number of the quantified variables in $\varphi \wedge \psi$ is fixed, then the size of $\mathsf{post}_\tau(\varphi)$ (resp. $\mathsf{pre}_\tau(\varphi)$), increases polynomially w.r.t. the size of the formula $\varphi \wedge \psi$.

Example 7.2. To illustrate the construction given in the proof above, we consider the logic $\mathsf{CML}(\mathbb{N}, \{0\}, \{\leq\})$ and the CPN $S = (\mathbb{P}, \Delta)$ with $\mathbb{P} = \{p, q, r\}$ and Δ containing the following transition:

$$\tau : \;\; p \hookrightarrow q \;:\; \delta_1(x_1) \geq 0 \wedge \neg(\exists t \in q.\, \delta_1(t) = \delta_1(y_1))$$

Intuitively, this transition moves a token with positive color from place p to place q and assigns to its color a value non-deterministically chosen in $\mathbb{N}$ but different from all colors of tokens in place q.

We illustrate the computation of post-image of τ on two formulas in special form $\varphi_1 = (\exists x \in r.\, \mathit{true})$ and $\varphi_2 = (\forall x, y \in p.\, x = y)$. Intuitively, φ_1 says that the place r contains at least a token, and φ_2 says that any two tokens in place p are equal, i.e., place p contains at most one token. Since φ_1 is not speaking about places involved in the transition τ (i.e., p and q), we expect to obtain a stable post-image by τ, i.e., $\varphi_{1,\mathsf{post}_\tau} \implies \varphi_1$. Conversely, φ_2 speaks about a place changed by τ, so its image cannot be stable. In the remainder of this example we give the details of the construction of the post-images by τ for φ_1 and φ_2.

By applying the equation 7.1 to φ_1 we obtain:

$$\begin{array}{rcl} \varphi_{1,\mathsf{post}_\tau} & = & \exists y_1 \in q.\, \exists c_{1,x_1}.\;\; (\varphi_1 \wedge \delta_1(x_1) \geq 0 \wedge \neg(\exists t \in q.\, \delta_1(t) = \delta_1(y_1)) \\ & & \qquad\qquad\qquad\quad) \ominus (\{x_1\}, \{x_1 \mapsto p\}, \{x_1 \mapsto 1 \mapsto c_{1,x_1}\}) \\ & & \qquad\qquad\qquad\quad \oplus (\{y_1\}, \{y_1 \mapsto q\}) \end{array}$$

In the following, we denote by $\mathtt{loc}_{x_1}$, $\mathtt{col}_{x_1}$, and $\mathtt{loc}_{y_1}$ the mappings $\{x_1 \mapsto p\}$, $\{x_1 \mapsto 1 \mapsto c_{1,x_1}\}$, resp. $\{y_1 \mapsto q\}$.

First, we compute the effect of applying the $\ominus$ operation on φ_1 and the guard of τ using the rules given in Table 3. By applying several times rules $\ominus_4$ and $\ominus_5$ to distribute $\ominus$ over

$\wedge$ and $\neg$ we obtain:

$$\begin{array}{rl} & \varphi_1 \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \\ \wedge & (\delta_1(x_1) \geq 0) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \\ \wedge & \neg\big((\exists t \in q.\ \delta_1(t) = \delta_1(y_1)) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1})\big) \end{array}$$

By applying rule $\ominus_6$ two times, $\ominus_2$ one time, and by replacing the empty disjunction by *false*, we obtain:

$$\begin{array}{rl} & \big(\exists x \in r.\ \mathit{true} \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \vee \mathit{false}\big) \\ \wedge & (c_{1,x_1} \geq 0) \\ \wedge & \neg\big(\exists t \in q.\ (\delta_1(t) = \delta_1(y_1)) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \vee \mathit{false}\big) \end{array}$$

Rules $\ominus_1$ and $\ominus_2$ are applied to obtain the final result:

$$\begin{array}{rl} & (\exists x \in r.\ \mathit{true}) \\ \wedge & (c_{1,x_1} \geq 0) \\ \wedge & \neg(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)) \end{array}$$

On the above formula is applied the $\oplus$ transformation using the rules given in Table 4. By applying several times rules $\oplus_4$ and $\oplus_5$ to distribute $\oplus$ over $\wedge$ and $\neg$, we obtain:

$$\begin{array}{rl} & (\exists x \in r.\ \mathit{true}) \oplus (\{y_1\}, \mathtt{loc}_{y_1}) \\ \wedge & (c_{1,x_1} \geq 0) \oplus (\{y_1\}, \mathtt{loc}_{y_1}) \\ \wedge & \neg\big((\exists t \in q.\ \delta_1(t) = \delta_1(y_1)) \oplus (\{y_1\}, \mathtt{loc}_{y_1})\big) \end{array}$$

By applying two times rules $\oplus_6$ and $\oplus_2$, and by replacing empty conjunctions by *true* we obtain:

$$\begin{array}{rl} & (\exists x \in r.\ \mathit{true} \oplus (\{y_1\}, \mathtt{loc}_{y_1})\ \wedge\ \mathit{true}) \\ \wedge & (c_{1,x_1} \geq 0) \\ \wedge & \neg\big(\exists t \in q.\ (\delta_1(t) = \delta_1(y_1)) \oplus (\{y_1\}, \mathtt{loc}_{y_1})\ \wedge\ \neg(t = y_1)\big) \end{array}$$

Rules $\oplus_1$ and $\oplus_2$ are applied to obtain the final result:

$$\begin{array}{rl} & (\exists x \in r.\ \mathit{true}) \\ \wedge & (c_{1,x_1} \geq 0) \\ \wedge & \neg\big(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)\ \wedge\ \neg(t = y_1)\big) \end{array}$$

Therefore, the immediate successors of φ_1 by τ are given by the following CML formula:

$$\begin{array}{l} \varphi_{1,\mathsf{post}_\tau} \\ \quad = \ \exists y_1 \in q.\ \exists c_{1,x_1}.\ (\exists x \in r.\ \mathit{true}) \wedge (c_{1,x_1} \geq 0) \wedge \neg\big(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)\ \wedge\ \neg(t = y_1)\big) \\ \quad = \ (\exists x \in r.\ \mathit{true}) \wedge \big(\exists y_1 \in q.\ \exists c_{1,x_1}.\ (c_{1,x_1} \geq 0) \wedge \neg\big(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)\ \wedge\ \neg(t = y_1)\big)\big) \end{array}$$

where the last equality has been obtained by applying classical rules for quantifiers. It is easy now to see that $\varphi_{1,\mathsf{post}_\tau} \implies \varphi_1$.

Now, we consider φ_2 and we apply the equation 7.1 to obtain:

$$\begin{array}{rcl} \varphi_{2,\mathsf{post}_\tau} & = & \exists y_1 \in q.\ \exists c_{1,x_1}.\ \ (\varphi_2 \wedge \delta_1(x_1) \geq 0 \wedge \neg(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)) \\ & & \qquad\qquad\qquad\ \ ) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \\ & & \qquad\qquad\qquad\quad \oplus (\{y_1\}, \mathtt{loc}_{y_1}) \end{array}$$

We only detail the effect of $\oplus$ and $\ominus$ operators on φ_2 since the computation for the conjunct representing the guard of τ is the same as for φ_1.

In order to apply $\ominus$ on φ_2, we use the equivalent form of φ_2, i.e., $\neg(\exists x \in p.\ \exists y \in p.\ \neg(x = y))$. Then, the effect of the $\ominus$ operation on φ_2 is obtained by applying two times the rules $\ominus_4$ and $\ominus_6$ as follows:

$$\begin{aligned}\varphi_2 \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \;=\; \neg\big((\exists x \in p.\ \ & (\exists y \in p.\ \neg(x = y) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1})) \\ & \vee\neg(x = x_1) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1})\ ) \\ \vee\ & (\exists y \in p.\ \neg(x_1 = y)) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1})\big)\end{aligned}$$

By applying several times rules $\ominus_3$, $\ominus_4$, and $\ominus_6$ we obtain:

$$\begin{aligned}&\varphi_2 \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \\ &\quad= \neg\big(\ (\exists x \in p.\ \exists y \in p.\ \neg(x = y) \vee \neg(\mathit{false})\ ) \\ &\qquad\quad \vee\ (\exists y \in p.\ \neg(x_1 = y) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}) \vee \neg(x_1 = x_1) \ominus (\{x_1\}, \mathtt{loc}_{x_1}, \mathtt{col}_{x_1}))\big) \\ &\quad= \neg\big(\ (\exists x \in p.\ \exists y \in p.\ \mathit{true}) \\ &\qquad\quad \vee\ (\exists y \in p.\ \neg(\mathit{false}) \vee \neg(\mathit{true}))\big) \\ &\quad= \neg(\exists x \in p.\ \exists y \in p.\ \mathit{true}) \wedge \neg(\exists y \in p.\ \mathit{true}) \\ &\quad= (\forall x, y \in p.\ \mathit{false}) \wedge (\forall y \in p.\ \mathit{false}) \\ &\quad= (\forall x \in p.\ \mathit{false})\end{aligned}$$

The last equivalence above is obtained from the classical properties of quantifiers. The final result is the one expected intuitively: the effect of removing the token x_1 in p from a configuration where there is at most one token in p (see meaning of φ_2) is a configuration with no token in p.

It is easy to show that the effect of $\oplus(\{y_1\}, \mathtt{loc}_{y_1})$ on the last formula above is null. Therefore, the immediate successors of φ_2 by τ are given by the following CML formula:

$$\begin{aligned}&\varphi_{2,\mathsf{post}_\tau} \\ &\quad= \exists y_1 \in q.\ \exists c_{1,x_1}.\ (\forall x \in p.\ \mathit{false}) \wedge (c_{1,x_1} \geq 0) \wedge \neg\big(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)\ \wedge\ \neg(t = y)\big) \\ &\quad= (\forall x \in p.\ \mathit{false}) \wedge \big(\exists y_1 \in q.\ \exists c_{1,x_1}.\ (c_{1,x_1} \geq 0) \wedge \neg\big(\exists t \in q.\ \delta_1(t) = \delta_1(y_1)\ \wedge\ \neg(t = y)\big)\big)\end{aligned}$$

More complex examples of post-image computations for the reader-writer lock example are provided in Section 9.1.

8. Applications in Verification

We show in this section how to use the results of the previous section to perform various kinds of analysis. Let us fix for the rest of the section a first order logic $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$ with a decidable satisfiability problem and a CPN S.

8.1. **Pre-post condition reasoning.** Given a transition τ in S and given two formulas φ and φ', $\langle \varphi, \tau, \varphi' \rangle$ is a Hoare triple if whenever the condition φ holds, the condition φ' holds after the execution of τ. In other words, we must have $\mathsf{post}_\tau([\![\varphi]\!]) \subseteq [\![\varphi']\!]$, or equivalently that $\mathsf{post}_\tau([\![\varphi]\!]) \cap [\![\neg\varphi']\!] = \emptyset$. Then, by Theorem 7.1 and Theorem 4.1 we deduce the following:

Theorem 8.1. *If S is a $\mathsf{CPN}[\Sigma_2]$, then the problem whether $\langle \varphi, \tau, \varphi' \rangle$ is a Hoare triple is decidable for every transition τ of S, every formula $\varphi \in \Sigma_2$, and every formula $\varphi' \in \Pi_2$.*

8.2. **Bounded reachability analysis.** An instance of the bounded reachability analysis problem is a triple $(Init, Target, k)$ where $Init$ and $Target$ are two sets of configurations, and k is a positive integer. The problem consists in deciding whether there exists a computation of length at most k which starts from some configuration in $Init$ and reaches a configuration in $Target$. In other words, the problem consists in deciding whether $Target \cap \bigcup_{0 \leq i \leq k} \mathsf{post}_S^i(Init) \neq \emptyset$, or equivalently whether $Init \cap \bigcup_{0 \leq i \leq k} \mathsf{pre}_S^i(Target) \neq \emptyset$. The following result is a direct consequence of Theorem 7.1 and Theorem 4.1.

Theorem 8.2. *If S is a $\mathsf{CPN}[\Sigma_2]$, then, for every $k \in \mathbb{N}$, and for every two formulas $\varphi_I, \varphi_T \in \Sigma_2$, the bounded reachability problem $([\![\varphi_I]\!], [\![\varphi_T]\!], k)$ is decidable.*

8.3. **Checking invariance properties.** Invariance checking consists in deciding whether a given property (1) is satisfied by the set of initial configurations, and (2) is stable under the transition relation of a system.

Formally, given a CPN S with transitions in Δ and a closed formula φ_{init} defining the set of initial configurations, we say that a closed formula φ is an *inductive invariant* of (Δ, φ_{init}) if and only if (1) $[\![\varphi_{init}]\!] \subseteq [\![\varphi]\!]$, and (2) $\mathsf{post}_\tau([\![\varphi]\!]) \subseteq [\![\varphi]\!]$ for any $\tau \in \Delta$. Clearly, (1) is equivalent to $[\![\varphi_{init}]\!] \cap [\![\neg\varphi]\!] = \emptyset$, and (2) is equivalent to $\mathsf{post}_\tau([\![\varphi]\!]) \cap [\![\neg\varphi]\!] = \emptyset$. By Theorem 7.1 and Theorem 4.1, we have:

Theorem 8.3. *The problem whether a formula $\varphi \in B(\Sigma_1)$ is an inductive invariant of (Δ, φ_{init}), where $\Delta \in \mathsf{CPN}[\Sigma_2]$ and $\varphi_{init} \in \Sigma_2$ is decidable.*

The deductive approach for establishing an invariance property considers the *inductive invariance checking problem* given by a triple $(\varphi_{init}, \varphi_{inv}, \varphi_{aux})$ of closed formulas expressing sets of configurations, and which consists in deciding whether (1) $[\![\varphi_{init}]\!] \subseteq [\![\varphi_{aux}]\!]$, (2) $[\![\varphi_{aux}]\!] \subseteq [\![\varphi_{inv}]\!]$, and (3) φ_{aux} is an inductive invariant. The following result is a direct consequence of Theorem 7.1, Theorem 4.1, and of the previous theorem.

Theorem 8.4. *If S is a $\mathsf{CPN}[\Sigma_2]$, then the inductive invariance checking problem is decidable for every instance $(\varphi_{init}, \varphi_{inv}, \varphi_{aux})$ where $\varphi_{init} \in \Sigma_2$, and $\varphi_{inv}, \varphi_{aux} \in B(\Sigma_1)$ are all closed formulas.*

Of course, the difficult part in applying the deductive approach is to find useful auxiliary inductive invariants. One approach to tackle this problem is to try to compute the largest inductive invariant included in φ_{inv} which is the set $\bigcap_{k \geq 0} \widetilde{\mathsf{pre}}_S^k(\varphi_{inv})$. Therefore, a method to derive auxiliary inductive invariants is to try iteratively the sets φ_{inv}, $\varphi_{inv} \cap \widetilde{\mathsf{pre}}_S(\varphi_{inv})$, $\varphi_{inv} \cap \widetilde{\mathsf{pre}}_S(\varphi_{inv}) \cap \widetilde{\mathsf{pre}}_S^2(\varphi_{inv})$, etc. In many practical cases, only few strengthening steps are needed to find an inductive invariant. (Indeed, the user is able in general to provide accurate invariant assertions for each control point of his system.) The result below implies that the steps of this iterative strengthening method can be automatized when $\mathsf{CPN}[\Sigma_1]$ models and Π_1 invariants are considered.

Theorem 8.5. *If S is a $\mathsf{CPN}[\Sigma_1]$, then for every closed formula φ in Π_1 and every positive integer k, it is possible to construct a formula in Π_1 defining the set $\bigcap_{0 \leq i \leq k} \widetilde{\mathsf{pre}}_S^i([\![\varphi]\!])$.*

The theorem above is a consequence of the fact that, by Theorem 7.1, for every S in $\mathsf{CPN}[\Sigma_1]$ and for every formula φ in Π_1, it is possible to construct a formula $\varphi_{\widetilde{\mathsf{pre}}}$ also in Π_1 such that $[\![\varphi_{\widetilde{\mathsf{pre}}}]\!] = \widetilde{\mathsf{pre}}_S([\![\varphi]\!])$.

Complexity: Let $\tau = \overrightarrow{p} \hookrightarrow \overrightarrow{q} : \psi$ be a transition of a system $S \in \mathsf{CPN}[\Sigma_2]$, and let φ be a $B(\Sigma_1)$ formula. The satisfiability of $\mathsf{post}_\tau(\varphi) \wedge \neg\varphi$ can be reduced in nondeterministic doubly-exponential time to the satisfiability problem of the color logic. This is due to the fact that (1) the reduction to the satisfiability problem of the color logic is in nondeterministic exponential time w.r.t. the maximal number of universally quantified variables in the formulas $\neg\varphi$ and $\mathsf{post}_\tau(\varphi)$, and that (2) the number of universally quantified variables in $\mathsf{post}_\tau(\varphi)$ is exponential in the number of universally quantified variables in $\varphi \wedge \psi$.

Now, for fixed sizes of $\overrightarrow{p}$ and $\overrightarrow{q}$, and for a fixed number of the quantified variables in $\varphi\wedge\psi$, the reduction to the satisfiability problem of the color logic is in NP. Such assumptions are in fact quite realistic in practice (as shown in the following section for different examples of parameterized systems). Indeed, in models of parametrized systems (see Section 6), communication involves only few processes (usually at most two). This justifies the bound on the sizes of left and right hand sides of the transition rules. Moreover, invariants are usually expressible using a small number of process indices (for instance mutual exclusion needs two indices) and relates only few of their local variables.

9. Case Studies and Experimental Results

We illustrate the use of our framework on several examples of parameterized systems. First, we consider the parameterized version of the Reader-Writer lock example provided in [FFQ02]. We give for this case study the inductive invariant allowing to prove a suitable safety property, and we show significant parts of its proof.

Then, we describe briefly a prototype tool for checking invariance properties based on our framework, and we give the experimental results obtained on several examples of parameterized mutual exclusion protocols and on the Reader-Writer lock case study.

9.1. Verification of the Reader-Writer Lock.

A safety property of our example is "for all Reader processes at control location 3, the local variable y has the same value, equal to f(x)", whose specification in CML is the following Π_1 formula:

$$RF = \forall a \in r3, t \in x.\ \delta_2(a) = f(\delta_1(t))$$

Of course, this property is true only if all Reader and Writer processes respect the procedure of acquiring the lock, i.e., there are no other processes in the system which are accessing the global variable x. Therefore, a correct initial configuration of the CPN model given on Table 6.2 has no token in places $r2$, $r3$, $w2$, and $w3$, and only one token in place x. Moreover, all process identities stored in color δ_1 are positive. We suppose that the lock is free initially, i.e., the place r is empty and the place w contains a unique token with negative δ_2 color. Then, a correct initial configuration of the system is given by the following Init formula in $B(\Sigma_1)$:

$$\mathit{Init} = G_x \wedge \mathit{Ids} \wedge \mathit{Init}_{lock} \wedge \Big(\forall t.\ \neg\big(r2(t) \vee r3(t) \vee w2(t) \vee w3(t)\big)\Big)$$

where

$$G_x = (\exists t \in x.\ \mathit{true}) \wedge (\forall t, t' \in x.\ t = t')$$

expresses that the place x contains a unique token,

$$\mathit{Ids} = \forall t.\ \delta_1(t) \geq 0$$

expresses that all tokens have a positive color δ_1 (representing their identity), and

$$Init_{lock} = (\exists u \in w.\ \delta_2(u) < 0) \wedge (\forall u, u' \in w.\ u = u') \wedge (\forall t \in r.\ false)$$

specifies the initial state of the lock: there is only one token in place w and its color δ_2 is negative, and the place r is empty.

The premises of Theorem 8.4 are fulfilled since the model proposed on Table 6.2 is in $\mathsf{CPN}[\Pi_1]$, and *Init* and *RF* are both in $B(\Sigma_1)$. It follows that we have to find an inductive invariant $\varphi_{aux} \in B(\Sigma_1)$ such that $Init \implies \varphi_{aux}$ and $\varphi_{aux} \implies RF$. We consider the following $B(\Sigma_1)$ formula as candidate for φ_{aux}:

$$Aux = G_x \wedge Ids \wedge RW_w \wedge RW_r \wedge RF$$

where G_x, *Ids* and *RF* are defined above and

$$RW_w = (\exists u \in w.\ true) \wedge (\forall u, u' \in w.\ u = u') \wedge ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0))$$

specifies that the place w contains only one token which color δ_2 is positive when a writer process is accessing the global variable (because δ_2 stores the identity of the writer), and

$$RW_r = (\exists v.\ r2(v) \vee r3(v)) \Leftrightarrow (\exists \ell \in r.\ true)$$

expresses that the place r must contain a token when a reader process is accessing the global variable (i.e., it is at locations $r2$ or $r3$).

Therefore, to check the safety property *RF* we have to show that: (1) $Init \implies Aux$, (2) for any transition τ in the system, $\mathsf{post}_\tau(Aux) \implies Aux$, and (3) $Aux \implies RF$. We let the point (1) as an exercise. The point (3) follows trivially from the definition of *Aux*. In the following, we detail the proof of the point (2) for one transition of the system, namely w_1, that we recall hereafter for readability:

$$\begin{array}{llllll} w_1: & w1, w & \hookrightarrow & w2, w & : & \neg(\exists z \in r.\ true) \wedge \delta_2(x_2) < 0 \wedge \delta_2(y_2) = \delta_1(x_1) \wedge \\ & & & & & \delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1) \end{array}$$

Using equation 7.1, we obtain that the post-image of *Aux* by the transition w_1 has the following form:

$$\begin{array}{l} Aux_{\mathsf{post}_{w_1}} \\ \quad = \ \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{1,x_1}, c_{2,x_1}, c_{1,x_2}, c_{2,x_2}. \\ \qquad\quad \big(\ (Aux \wedge \neg(\exists z \in r.\ true) \wedge \\ \qquad\qquad\ \delta_2(x_2) < 0 \wedge \delta_2(y_2) = \delta_1(x_1) \wedge \delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1) \\ \qquad\qquad\ ) \ominus (\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\ \qquad\quad \big) \oplus (\overrightarrow{y}, \mathtt{loc}_{\overrightarrow{y}}) \end{array}$$

where $\overrightarrow{x} = (x_1, x_2)$, $\mathtt{loc}_{\overrightarrow{x}} = [x_1 \mapsto w1, x_2 \mapsto w]$, $\mathtt{col}_{\overrightarrow{x}} = [x_i \mapsto k \mapsto c_{k,x_i}]_{1 \leq i \leq 2, 1 \leq k \leq 2}$, $\overrightarrow{y} = (y_1, y_2)$, and $\mathtt{loc}_{\overrightarrow{y}} = [y_1 \mapsto w2, y_2 \mapsto w]$.

Before applying operators $\ominus$ and $\oplus$, let us observe that *Aux*'s closed sub-formulas G_x, RW_r, *RF*, and $\neg(\exists z \in r.\ true)$ concern places which are not involved in the transition w_1. It can be shown that (and Example 7.2 gives an illustration of this fact) these sub-formulas are not changed by the application of $\ominus$ and $\oplus$ operators. Therefore, we have to apply these operators only on the rest of sub-formulas of Aux and on the guard of w_1, i.e.:

$$
\begin{array}{l}
Aux_{\mathsf{post}_{w_1}} \\
\quad = \; G_x \wedge RW_r \wedge RF \wedge \neg(\exists z \in r.\ \mathit{true}) \wedge \\
\qquad \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{1,x_1}, c_{2,x_1}, c_{1,x_2}, c_{2,x_2}. \\
\qquad\quad \big(\ (\mathit{Ids} \wedge RW_w \wedge \\
\qquad\qquad \delta_2(x_2) < 0 \wedge \delta_2(y_2) = \delta_1(x_1) \wedge \delta_1(y_2) = \delta_1(x_2) \wedge \varphi_{id}(1) \\
\qquad\qquad)\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
\qquad\quad \big)\oplus(\overrightarrow{y}, \mathtt{loc}_{\overrightarrow{y}})
\end{array}
$$

By distributing the $\ominus$ operator over $\wedge$ (rules $\ominus_4$ and $\ominus_5$), and by applying three times the rule $\ominus_2$, we obtain:

$$
\begin{array}{l}
Aux_{\mathsf{post}_{w_1}} \\
\quad = \; G_x \wedge RW_r \wedge RF \wedge \neg(\exists z \in r.\ \mathit{true}) \wedge \\
\qquad \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{1,x_1}, c_{2,x_1}, c_{1,x_2}, c_{2,x_2}. \\
\qquad\quad \big(\ (\mathit{Ids}\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \wedge RW_w \ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}})\wedge \\
\qquad\qquad c_{2,x_2} < 0 \wedge \delta_2(y_2) = c_{1,x_1} \wedge \delta_1(y_2) = c_{1,x_2} \wedge \varphi_{id}(1)\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
\qquad\quad \big)\oplus(\overrightarrow{y}, \mathtt{loc}_{\overrightarrow{y}})
\end{array}
$$

The application of $\ominus$ on the Ids sub-formula uses the rules $\ominus_4$ and $\ominus_6$ and has as effect the introduction of constraints on the c_{1,x_1} and c_{2,x_1} color variables:

$$
\begin{array}{rcl}
\mathit{Ids}\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) & = & (\forall t.\ \delta_1(t) \geq 0)\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
 & = & \mathit{Ids} \wedge c_{1,x_1} \geq 0 \wedge c_{1,x_2} \geq 0
\end{array}
$$

The result of applying $\ominus$ on the RW_w sub-formula is (sometimes we omit the arguments of $\ominus$ for legibility):

$$
\begin{array}{l}
RW_w \ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
\quad = \; \big(\ (\exists u \in w.\ \mathit{true})\wedge \\
\qquad\quad (\forall u, u' \in w.\ u = u')\wedge \\
\qquad\quad ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0))\big)\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
\quad = \; (\exists u \in w.\ \mathit{true})\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \wedge \\
\qquad (\forall u, u' \in w.\ u = u')\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \wedge \\
\qquad ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0))\ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
\quad = \; ((\exists u \in w.\ \mathit{true}) \vee \mathit{true}) \wedge \\
\qquad (\forall u \in w.\ (\forall u' \in w.\ (u = u')\ominus) \wedge (u = x_2)\ominus) \wedge (\forall u' \in w.\ (x_2 = u')\ominus) \wedge (x_2 = x_2)\ominus \wedge \\
\qquad ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0 \vee c_{2,x_2} \geq 0)) \\
\quad = \; \mathit{true} \wedge \\
\qquad (\forall u \in w.\ (\forall u' \in w.\ u = u') \wedge \mathit{false}) \wedge (\forall u' \in w.\ \mathit{false}) \wedge \mathit{true} \wedge \\
\qquad ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0 \vee c_{2,x_2} \geq 0))
\end{array}
$$

After some trivial simplification, we obtain:

$$
\begin{array}{rcl}
RW_w \ominus(\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) & = & (\forall u \in w.\ \mathit{false}) \wedge \\
 & & ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0 \vee c_{2,x_2} \geq 0))
\end{array}
$$

As expected, the first conjunct of the result obtained above says that after the deletion of the x_2 token in w, there is no more token in w.

The result of applying $\ominus$ on the $\varphi_{id}(1)$ sub-formula is:

$$\begin{aligned}
\varphi_{id}(1) \ominus (\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) &= (\delta_1(x_1) = \delta_1(y_1) \wedge \delta_2(x_1) = \delta_2(y_1)) \ominus (\overrightarrow{x}, \mathtt{loc}_{\overrightarrow{x}}, \mathtt{col}_{\overrightarrow{x}}) \\
&= \delta_1(y_1) = c_{1,x_1} \wedge \delta_2(y_1) = c_{2,x_1}
\end{aligned}$$

Therefore, after applying the $\ominus$ operator we obtain:

$$\begin{aligned}
& Aux_{\mathsf{post}_{w_1}} \\
&\quad = G_x \wedge RW_r \wedge RF \wedge \neg(\exists z \in r.\ true) \wedge \\
&\qquad \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{1,x_1}, c_{2,x_1}, c_{1,x_2}, c_{2,x_2}. \\
&\qquad\quad \big(\ (Ids \wedge c_{1,x_1} \geq 0 \wedge c_{1,x_2} \geq 0 \wedge (\forall u \in w.\ false) \wedge \\
&\qquad\quad\ \ ((\exists t.\ w2(t) \vee w3(t)) \Leftrightarrow (\exists u \in w.\ \delta_2(u) \geq 0 \vee c_{2,x_2} \geq 0)) \wedge \\
&\qquad\quad\ \ c_{2,x_2} < 0 \wedge \delta_2(y_2) = c_{1,x_1} \wedge \delta_1(y_2) = c_{1,x_2} \wedge \delta_1(y_1) = c_{1,x_1} \wedge \delta_2(y_1) = c_{2,x_1} \\
&\qquad\quad \big) \oplus (\overrightarrow{y}, \mathtt{loc}_{\overrightarrow{y}})
\end{aligned}$$

The $\oplus$ operation transforms all sub-formulas containing quantifiers. Indeed, after distributing $\oplus$ over conjunctions (rules $\oplus_4$ and $\oplus_5$) and after applying several times rules $\oplus_2$ and $\oplus_6$, we obtain:

$$\begin{aligned}
& Aux_{\mathsf{post}_{w_1}} \\
&\quad = G_x \wedge RW_r \wedge RF \wedge \neg(\exists z \in r.\ true) \wedge \\
&\qquad \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{1,x_1}, c_{2,x_1}, c_{1,x_2}, c_{2,x_2}. \\
&\qquad\quad \big(\ (\forall t.\ \delta_1(t) \geq 0 \vee (t = y_1) \vee (t = y_2)) \wedge c_{1,x_1} \geq 0 \wedge c_{1,x_2} \geq 0 \wedge \\
&\qquad\quad\ \ (\forall u \in w.\ false \vee u = y_2) \wedge \\
&\qquad\quad\ \ \big((\exists t.\ (w2(t) \vee w3(t)) \wedge (t \neq y_1)) \Leftrightarrow ((\exists u \in w.\ \delta_2(u) \geq 0 \wedge (u \neq y_2)) \vee c_{2,x_2} \geq 0)\big) \wedge \\
&\qquad\quad\ \ c_{2,x_2} < 0 \wedge \delta_2(y_2) = c_{1,x_1} \wedge \delta_1(y_2) = c_{1,x_2} \wedge \delta_1(y_1) = c_{1,x_1} \wedge \delta_2(y_1) = c_{2,x_1} \\
&\qquad\quad \big)
\end{aligned}$$

We can now apply the decision procedure defined in Section 4 to prove that $Aux_{\mathsf{post}_{w_1}} \Longrightarrow Aux$, i.e., $Aux_{\mathsf{post}_{w_1}} \wedge \neg Aux$ is unsatisfiable. Instead of doing this proof, we give some hints about the validity of this implication. First, we remark that by projecting color variables c_{1,x_1} and c_{1,x_2} the *Ids* sub-formula of *Aux* is implied by the sub-formula $(\forall t.\ \delta_1(t) \geq 0 \vee (t = y_1) \vee (t = y_2))$ and the constraints on $\delta_1(y_1)$ and $\delta_1(y_2)$:

$$\begin{aligned}
& Aux_{\mathsf{post}_{w_1}} \\
&\quad = G_x \wedge RW_r \wedge RF \wedge \neg(\exists z \in r.\ true) \wedge \\
&\qquad \exists y_1 \in w2.\ \exists y_2 \in w.\ \exists c_{2,x_1}, c_{2,x_2}. \\
&\qquad\quad \big(\ (\forall t.\ \delta_1(t) \geq 0 \vee (t = y_1) \vee (t = y_2)) \wedge \delta_1(y_1) \geq 0 \wedge \delta_1(y_2) \geq 0 \wedge \\
&\qquad\quad\ \ (\forall u \in w.\ u = y_2) \wedge \\
&\qquad\quad\ \ \big(\Leftrightarrow ((\exists u \in w.\ \delta_2(u) \geq 0 \wedge (u \neq y_2)) \vee c_{2,x_2} \geq 0)\big) \wedge \\
&\qquad\quad\ \ c_{2,x_2} < 0 \wedge \delta_2(y_2) \geq 0 \wedge \delta_2(y_1) = c_{2,x_1} \\
&\qquad\quad \big)
\end{aligned}$$

Second, RW_w sub-formula of *Aux* is implied by the sub-formula $\exists y_1 \in w_2.\ \exists y_2 \in w.\ \ldots (\forall u \in w.\ u = y_2) \wedge \cdots \wedge \delta_2(y_2) \geq 0 \ldots$. Finally, in the context of conjuncts $c_{2,x_2} < 0$ and

Algorithm	*Nb. rules*	*Inv. size*	*SMT Lemmas*	*Time (sec.)*
Burns [BL80]	9	6	92	0.81
Ticket	3	9	28	26.23
Bakery [Lam74]	3	5	10	0.15
Dijkstra [Dij65]	11	9	1177	18390.97
Martin [Mar86]	8	7	837	980.97
Szymanski [Szy88]	9	12	293	1065.1
Reader-writer lock [FFQ02]	6	9	70	2195.68

Table 5: Experimental results.

$(\forall u \in w.\ u = y_2)$, the left member of the equivalence:

$$\big((\exists t.\ (w2(t) \vee w3(t)) \wedge (t \neq y_1)) \Leftrightarrow ((\exists u \in w.\ \delta_2(u) \geq 0 \wedge (u \neq y_2)) \vee c_{2,x_2} \geq 0)\big)$$

is false, so we can replace it by $\neg(\exists t.\ (w2(t) \vee w3(t)) \wedge (t \neq y_1))$ which expresses, as expected, that only one writer (here y_1) can be present at the location $w2$.

9.2. **Experimental results.** We have implemented the algorithms for the decision procedure of CML, the post and pre-image computations, and the inductive invariant checking.

Our prototype tool, implemented in Ocaml, takes as input an invariant φ_{inv} in $B(\Sigma_1)$ which is a conjunction of local invariants written in special form (see definition in Section 2.3.3). Indeed, the invariants are usually conjunctions of formulas, each of them being an assertion which must hold when the control is at some particular location. Then, it decomposes the inductive invariant checking problem (i.e., $\mathsf{post}(\varphi_{inv}) \wedge \neg\varphi_{inv}$ is unsatisfiable) in several lemmas, one lemma for each transition of the input CPN model and for each local invariant in φ_{inv} which contains places involved in the transition. For example, the tool generates 70 lemmas for the verification of the inductive invariant for the *RF* property on the Reader-Writer lock example. However, not all lemmas are generated if the decision procedure for CML returns satisfiable for one of them (which implies that φ_{inv} is not an inductive invariant). The implemented decision procedure for CML is parameterized by the decision procedure for the logic of colors $\mathsf{FO}(\mathbb{C}, \Omega, \Xi)$. Actually, we generate lemmas in the SMTLIB format and we have an interface with most known SMT solvers. Therefore, we can allow as color logic any theory supported by the state of the art SMT solvers.

Using this prototype, we modeled and verified several parameterized versions for mutual exclusion algorithms. The experimental results are given on Table 5. (The considered models of the Burns and Bakery algorithms use atomic global condition checks over all the processes, although our framework allows in principle the consideration of models where global conditions are checked using non atomic iterations over the set of processes.) For all these examples, the color logic is the difference logic over integers for which we have used the decision procedure of Yices [DdM06]. For each example, Table 5 gives the number of rules of the model, the number of conjuncts of the inductive invariant (in CNF), the number of lemmas generated for the SMT solver, and the global execution time.

10. Conclusion

We have presented a framework for reasoning about dynamic/parametric networks of processes manipulating data over infinite domains. We have provided generic models for these systems and a logic allowing to specify their configurations, both being parametrized by a logic on the considered data domain. We have identified a fragment of this logic having a decidable satisfiability problem and which is closed under post and pre image computation, and we have shown the application of these results in verification.

Our framework allows to deal in a uniform way with all classes of systems manipulating infinite data domains with a decidable first-order theory. In this paper, we have considered instantiations of this framework based on logics over integers or reals (which allows to consider systems with numerical variables). Different data domains can be considered in order to deal with other classes of systems such as multithreaded programs where each process (thread) has an unbounded stack (due to procedure calls). Our future work includes also the extension of our framework to other classes of systems and features such as dynamic networks of timed processes, networks of processes with broadcast communication, interruptions and exception handling, etc.

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proceedings of CAV*, volume 1855 of *LNCS*, pages 419–434. Springer-Verlag, July 2000.

[AD06] P. A. Abdulla and G. Delzanno. On the Coverability Problem for Constrained Multiset Rewriting. In *Proceedings of AVIS, Satellite workshop of ETAPS*, 2006.

[ADHR07] P. A. Abdulla, G. Delzanno, N. Ben Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *Proceedings of TACAS*, volume 4424 of *LNCS*, pages 721–736. Springer-Verlag, 2007.

[AHDR08] P. A. Abdulla, N. Ben Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *Proceedings of VMCAI*, volume 4905 of *LNCS*, pages 22–36. Springer-Verlag, 2008.

[AJ98] P. A. Abdulla and B. Jonsson. Verifying networks of timed processes (extended abstract). In *Proceedings of TACAS*, volume 1384 of *LNCS*, pages 298–312. Springer-Verlag, 1998.

[AJNS04] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A Survey of Regular Model Checking. In *Proceedings of CONCUR*, volume 3170 of *LNCS*. Springer-Verlag, 2004.

[APR$^+$01] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proceedings of CAV*, volume 2102 of *LNCS*. Springer-Verlag, 2001.

[Ari05] M. Ben Ari. *Principle of Concurrent and Distributed Programming.* Addison-Wesley, 2005.

[AvJT96] P. A. Abdulla, K. Čerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of LICS*, pages 313–321. IEEE, 1996.

[BD02] M. Bozzano and G. Delzanno. Beyond Parameterized Verification. In *Proceedings of TACAS*, volume 2280 of *LNCS*. Springer-Verlag, 2002.

[BDM$^+$06] M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *Proceedings of PODS*. ACM, 2006.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, 1997.

[BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *Proceedings of CAV*, volume 3114 of *LNCS*. Springer-Verlag, 2004.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In *Proceedings of CAV*, volume 1855 of *LNCS*. Springer-Verlag, 2000.

[BL80] J. Burns and N.A. Lynch. Mutual exclusion using indivisible reads and writes. In *Proceedings of the 18th Allerton Conference on Communication, Control and Computing*, pages 833–842, October 1980.

[BMOT05] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proceedings of CONCUR*, volume 3653 of *LNCS*. Springer-Verlag, 2005.

[BMS$^+$06a] M. Bojanczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proceedings of LICS*. IEEE, 2006.

[BMS06b] A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays ? In *Proceedings of VMCAI*, volume 3855 of *LNCS*. Springer-Verlag, 2006.

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Space*. PhD thesis, Faculté des Sciences, Université de Liège, volume 189, 1999.

[Bou01] A. Bouajjani. Languages, Rewriting systems, and Verification of Infinte-State Systems. In *Proceedings of ICALP*, volume 2076 of *LNCS*. Springer-Verlag, 2001.

[BT05] A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In *Proceedings of RTA*, volume 3467 of *LNCS*. Springer-Verlag, 2005.

[CGJ97] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks. *ACM Trans. Program. Lang. Syst.*, 19(5):726–750, 1997.

[DdM06] B. Dutertre and L. Mendonça de Moura. A fast linear-arithmetic solver for dpll(t). In *Proceedings of CAV*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.

[Del01] G. Delzanno. An assertional language for the verification of systems parametric in several dimensions. *Electr. Notes Theor. Comput. Sci.*, 50(4), 2001.

[Dij65] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of ACM*, 8(9):569, 1965.

[DL06] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *Proceedings of LICS*. IEEE, 2006.

[DRB02] G. Delzanno, J.-F. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *Proceedings of TACAS*, volume 2280 of *LNCS*, pages 173–187. Springer-Verlag, 2002.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proceedings of LICS*, pages 352–359. IEEE, 1999.

[EN98] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proceedings of LICS*. IEEE, 1998.

[FFQ02] C. Flanagan, S.N. Freund, and S. Qadeer. Thread-modular verification for shared-memory programs. In *ESOP*, pages 262–277, 2002.

[FL02] A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proceedings of FST&TCS*, volume 2556 of *LNCS*. Springer-Verlag, 2002.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[GS92] S. M. German and P. A. Sistla. Reasoning about systems with many processes. *JACM*, 39(3), 1992.

[Lam74] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of ACM*, 17(8):453–455, August 1974.

[Mar86] A. J. Martin. A new generalization of Dekker's algorithm for mutual exclusion. *Inf. Process. Lett.*, 23(6):295–297, 1986.

[Szy88] B.K. Szymanski. A simple solution to Lamport's concurrent programming problem with linear wait. In *ICS*, pages 621–626, 1988.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proceedings of CAV*, volume 1427 of *LNCS*. Springer-Verlag, 1998.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Proceedings Intern. Workshop on Automatic Verification Methods for Finite State Systems*. LNCS 407, 1989.